NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

WIRELESS TOOL KIT FOR HAND HELD DEVICES

by

Venkateshwaraiyer S Baalaji

September 2004

Thesis Advisor:	Gurminder Singh
Thesis Co-Advisor:	Alex Bordetsky

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Wireless Tool Kit For Hand Held Devices			**5. FUNDING NUMBERS**
6. AUTHOR(S) Venkateshwaraiyer S Baalaji			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Wireless networks can be broadly classified into two types – infrastructure based networks and ad hoc networks. The former uses fixed base stations (infrastructure) which are responsible for coordinating communication between the mobile hosts (nodes). These base stations are interconnected by wired back bones, where as mobile nodes communicate with the base station through the wireless media. The latter one consists of mobile nodes which communicate with each other through wireless medium without any fixed infrastructure. Hence there is no centralized infrastructure that takes care of the routing of information among the participants in the network. There has been a growing interest in ad hoc network in recent years as mobile devices have become more powerful and are capable of processing data like their desktop counterparts. When such a capability is available these devices should be able to share information among them without reliance on existing network infrastructure. Mobile devices are self configurable into either infrastructure or ad hoc mode. Protocols and software have been developed to enable mobile devices to connect to an infrastructure node, where as the same is not the case in the ad hoc environment. Host mobility causes frequent and unpredictable topological changes in a wireless environment. Finding and maintaining routes in ad hoc networks is a non trivial task. This thesis will develop software components that will enable communication in an ad hoc network. These components could be used to build collaborative services in such and ad hoc (802.11) wireless environment.

14. SUBJECT TERMS Media Access control, Ad hoc routing protocol, Ad hoc network performance requirements.			**15. NUMBER OF PAGES** 93
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

WIRELESS TOOL KIT FOR HAND HELD DEVICES

Venkateshwaraiyer S Baalaji
Major, Indian Air Force
B.E. Computer Science, Madurai Kamaraj University, 1991.

Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2004

Author: Venkateshwaraiyer S Baalaji

Approved by: Dr Gurminder Singh
Thesis Advisor

Dr Alex Bordetsky
Co-Advisor

Dr Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Wireless networks can be broadly classified into two types – infrastructure based and ad hoc. The former uses fixed base stations (infrastructure) which are responsible for coordinating communication between the mobile hosts (nodes). These base stations are usually interconnected by wired backbones, where as mobile nodes communicate with the base station through the wireless media. Ad-hoc wireless networks consist of mobile nodes which communicate with each other through wireless medium without any fixed infrastructure. Hence there is no centralized infrastructure that takes care of the routing of information among the participants in the network.

There has been a growing interest in ad hoc network in recent years as mobile devices have become more powerful and capable of processing data like their desktop counterparts. When such a capability is available these devices should be able to share information among them without reliance on existing network infrastructure. Mobile devices are self configurable into either infrastructure or ad hoc mode. Protocols and software have been developed to enable mobile devices to connect to an infrastructure node, whereas it is not the same in the ad hoc environment. Host mobility causes frequent and unpredictable topological changes in a wireless environment. Finding and maintaining routes in ad hoc networks is a non trivial task. This thesis will develop software components that will enable communication in an ad hoc network. These components could be used to develop collaborative services in such an ad hoc (802.11) wireless environment.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AODV	-	Ad hoc On-Demand Distance Vector Routing
CSGR	-	Cluster Switch Gateway Routing
DBTMA	-	Dual Busy Tone Multiple Access
DSDV	-	Destination Sequenced Distance Vector
DSR	-	Dynamic Source Routing
EED	-	End to End Delay
LAR	-	Location Aided Routing
MAC	-	Media Access Protocol
MACA	-	Multiple Access with Collision Avoidance
MACA-BI	-	Multiple Access with Collision Avoidance (By Invitation)
MARCH	-	Media Access with Reduced Handshake
MANET	-	Mobile Ad hoc Network
PAMAS	-	Power Aware Multiple Access Protocol with Signaling
PAR	-	Power Aware Routing
PRNET	-	Packet Radio Networks
RD	-	Route Discovery
SSR	-	Signal Stability Routing
TORA	-	Temporally Ordered Routing Algorithm
WRP	-	Wireless Routing Protocol
ZRP	-	Zone Routing Protocol

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professor Gurminder Singh for his advice and guidance through out the process of development of the toolkit which led to the successful completion of the thesis. The genesis of the thesis was based on the requirement of Professor Gurminder Singh who has specialized in the area of wireless technologies.

I would also like to thank Professor Alex Bordetsky for having provided valuable guidance in completing the thesis.

I would like to thank my wife Anitha for her patience and understanding. This effort would not have been possible without her support. I would like to thank her for her effort in proof reading and correcting my language without which this effort would have been infinitely harder.

Finally to my daughter Abirami, my hope is that this achievement will pale in comparison to the success you will enjoy in your future endeavors.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Beginning in the early 1990s, engineers began to develop a networking standard that would allow users freedom of movement by employing the use of wireless communications. Today, that standard is established and the implementation of Wireless Local Area Networks (WLANs) is growing at an almost exponential rate.

Analysts from IDC Technologies project that global revenue for WLAN equipment will increase to $3.72 billion in 2006 from $1.45 billion in 2001 [1]. Most of the development has focused on the ability to provide users with the advantages of increased mobility, convenience of use and improved productivity. The WLANs have been built as an extension to the existing infrastructure that provides the additional facility of mobility to the users. All this has become possible with the advent of small yet powerful handheld/portable devices that have the capability to communicate with existing infrastructure nodes in a mobile environment. However, the possibility of these devices communicating among themselves without the use of the infrastructure nodes and providing services has not been exploited to its full extent. Such a network wherein the mobile devices communicate among themselves without reliance on the existing infrastructure is called an ad hoc network.

A wireless mobile ad hoc network (MANET) consists of a collection of peer mobile nodes that are capable of communicating with each other without help from a fixed infrastructure. The interconnections between these nodes are capable of changing in a continuous and arbitrary fashion. Nodes within each other's radio range communicate directly via wireless links while those that are far apart use other nodes as relays. Nodes usually share the same physical media: they transmit and acquire signals in the same frequency band, and follow the same hopping sequence or spreading code. The data link layer function manages the wireless link resources and coordinates medium access among neighboring nodes. The network layer function maintains the multi-hop communication path across the network: all nodes must function as routers that discover and maintain routes to other nodes in the network. Mobility and volatility are hidden from the applications so that any node can communicate with any other node as if

everyone were in the fixed wired network. There are many applications of ad hoc networks that range from military tactical operations to civil rapid deployment such as emergency search and rescue missions, data collection/sensor network and instantaneous classroom/meeting room applications. Currently, almost all PDAs available commercially come equipped with several different wireless networking options including 802.11, infrared and Bluetooth. The Operating Systems running on these devices provide support for software development in the infrastructure and ad hoc modes. However, the application development and support is available more in the infrastructure mode than in ad hoc. This thesis will develop a toolkit that will hide the network layer functionality required to build MANET application and help programmers to develop services in a MANET. MANET is a complex environment and protocols are being developed to address issues concerning the ad hoc nature of the network.

B. OBJECTIVE

The main objective of this thesis is to implement a software toolkit to facilitate the development of applications on ad hoc 802.11 wireless networks. There are many routing protocols that have been proposed for the set up of an ad hoc network. We will investigate the proposals and adapt one that is implement-able using existing software classes. The devices that operate in an Ad hoc network environment are expected to be "aware" of other devices operating in the environment and also aware of existing infrastructure around them (if any). This thesis shall develop an approach to making devices "aware" of other devices and network elements.

C. BENEFITS

The existence of mobile nodes that are capable of communicating among them in an ad hoc fashion increases the survival capacity of a network which is under hostile action. The capability of the network to adapt and transform itself also makes the network more difficult to attack. Clearly, the benefits that accrue from such a network are many fold. The ad hoc network elements would be aware of the existence of other nodes in the environment and also be able to route traffic destined to other nodes. Such an

environment would allow multiple paths of communication be formed as and when required. However, these nodes would have to be managed efficiently. The requirement in such a scenario would be the efficient use of bandwidth available and the effective establishment of routes from one node to another to carry information.

D. THESIS ORGANIZATION

Chapter II will illustrate the origin of ad hoc networking and the techniques adapted in setting up an ad hoc network using packet radios.

Chapter III will introduce the various wireless media access protocols that are currently employed. The thesis will use the existing IEEE 802.11 standard in its implementation and therefore discuss the employability of this standard in an ad hoc network.

Chapter IV will introduce the various wireless Ad-hoc routing algorithms that are evolving and being studied in detail. This chapter will take a look at the efficiencies and constraints of a few of the protocols.

Chapter V will emphasize the communication and performance issues. Performance issues relate to energy efficiency practices to be adapted in an ad hoc network so as to compensate for deficiencies in the battery technology that have caught up with speed of development of micro-processor technology.

Chapter VI will describe the software methodology followed to develop the toolkit.

Chapter VII will enumerate the lesson learnt during the implementation of the software and will list scope for future developments.

THIS PAGE INTENTIONALLY LEFT BLANK

II. AD HOC NETWORK

A. INTRODUCTION

The origin of an infrastructure less network dates back to the early 70s where computers and radio trans-receivers were bulky. DARPA had used several wireless terminals that could communicate with one another. The communication was based on the transmission of packets using packet radio technology and the concept of packet switching. ALOHA, developed at the University of Hawaii, showed the feasibility of broadcasting data packets in a single radio hop system. This single hop system was later upgraded to a multi hop communication system over a wide geographical area and was called the PRNET. The system consisted of mobile radio repeaters, wireless terminals and dedicated mobile stations. Repeaters relayed packets from one to another until the packet reached the destination. Mobile stations acted as routers deriving routes for the packets based on terminal movement, repeater failures, network congestion state and changes in hop reliability. Hosts and terminals joined the PRNET and were unaware of their stations assignments and reassignment in the network.

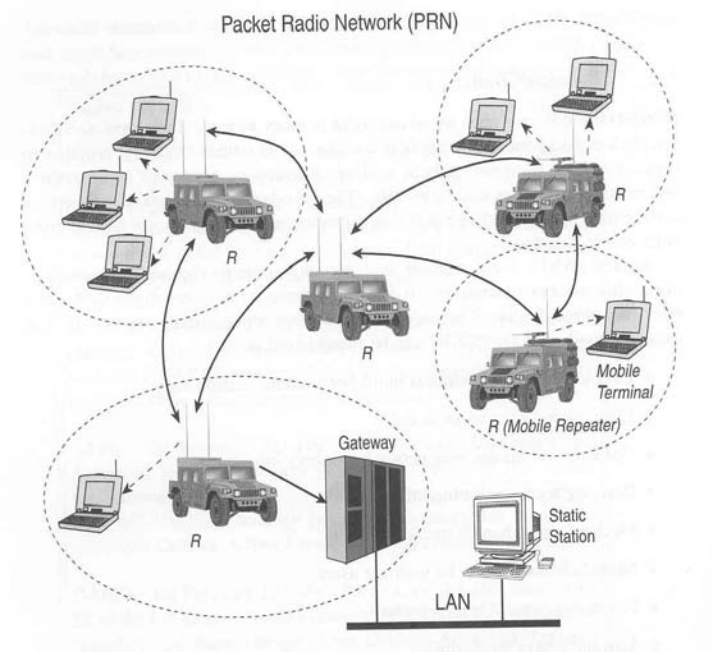

Figure 1. Network Architecture of PRNET. Static Station is Optional (From [2])

B. PRNET (PACKET RADIO NETWORKS – ORIGIN OF AD HOC NETWORKS)

PRNET allowed routing in both point to point and broadcast modes. Route calculations were based on tables. There were three tables that were monitored viz., neighbor table, tier table and device table. The neighbor table consisted of information regarding the existence of another radio device in its vicinity. This information was broadcast when a radio device was powered on in the form of a packet radio organization packet (PROP). The PROP was broadcast every 7.5secs which indicated the existence of other radio devices and also provided information about the network topology at that time. The bidirectional quality of the link to and from the neighbors was also stored. The link quality was measured as the number of packets correctly received from a transmitting station during a PROP interval to the number of packets that the station actually transmitted in the same interval.

Neighboring PR	Link Quality
Node 1	5/8
Node 7	1/3
Node 3	3/4
Node 5	7/8

Table 1. Structure of a Neighbor Table in PRN (From [2]).

The tier table consisted of information regarding the best route to be used to forward packets to every prospective destination. This information rippled outward from each packet radio at an average of 3.75secs. The best route was considered the shortest route with good connectivity. When a link turned bad, all routes using the link as the next hop were marked bad and this was also propagated via PROPS.

Destination PR	Next Hop PR	Tier Count
Node 1	Node 15	2
Node 7	Node 7	0
Node 3	Node 4	1
Node 5	Node 2	1

Table 2. Structure of Tier Table in PRN (From [2]).

The device table was used to maintain device to packet radio mapping. Each mobile device or terminal periodically sent control packets across the wired interface to its attached packet radio and this information was broadcast at an average rate of 3.75secs per hop.

The ETE (end to end) header used to transmit the radio packets included the source device ID/address and the destination device ID/address. This information remained intact as the packet moved from the source to its destination. In contrast to this header (ETE), another header called the *routing header* was created by the source packet radio. This header encapsulated the ETE header (which had to remain intact through out the packets journey) with relevant next hop information and acknowledgement fields. These fields were updated by the intermediate radio elements until the packet was successfully delivered. When delivered the routing header was discarded.

Routing Header Field	Purpose
Source PR ID	Acknowledgement
Sequence Number	Acknowledgement
Previous PR ID	Acknowledgement
Previous PR's transmit count	Pacing
Transmitting PR ID	Acknowledgement
Transmitting PR's transmit count	Pacing
Next PR ID	Acknowledgement/Forwarding-Pacing
Tier	Alternate Routing
Destination PR ID	Forwarding

Table 3. Structure of Routing Header (From [2]).

The time for transmission of these packets was based on a pacing protocol that provided flow and congestion control. The protocol required that transmitted packets be acknowledged before subsequent packets are sent. In addition only two buffers were provided per packet radio to accept packages received from the wired interface. Time delays were recorded between when a transmission is completed and the reception of the acknowledgement from the next packet radio. This was called the forwarding delay. Therefore the source packet radio had to wait for a three frame period in order to confirm that its packet had been sent, received at the intermediate node and then get acknowledgement of the forwarding of the same packet by the intermediate node. This ensured efficient use of the channel when there is no bottle neck and provided some congestion control when bottle necks were experienced.

Packet radios employed CSMA (carrier sense multiple access) for media access. The packet radio will not transmit when a carrier is sensed as this indicates transmission by another packet radio. Packets are forwarded in a single communication route. Each packet radio decides whether it is part of the route and transmits/forwards the packet only if it is and discards the packet if it isn't. If the packet radio is part of the route then it

forwards the packet and also provides a “passive” acknowledgement of the same to its previous hop. This continues until the packet is received at the node which does not have a downstream node or the destination which then does an “active” acknowledgement.

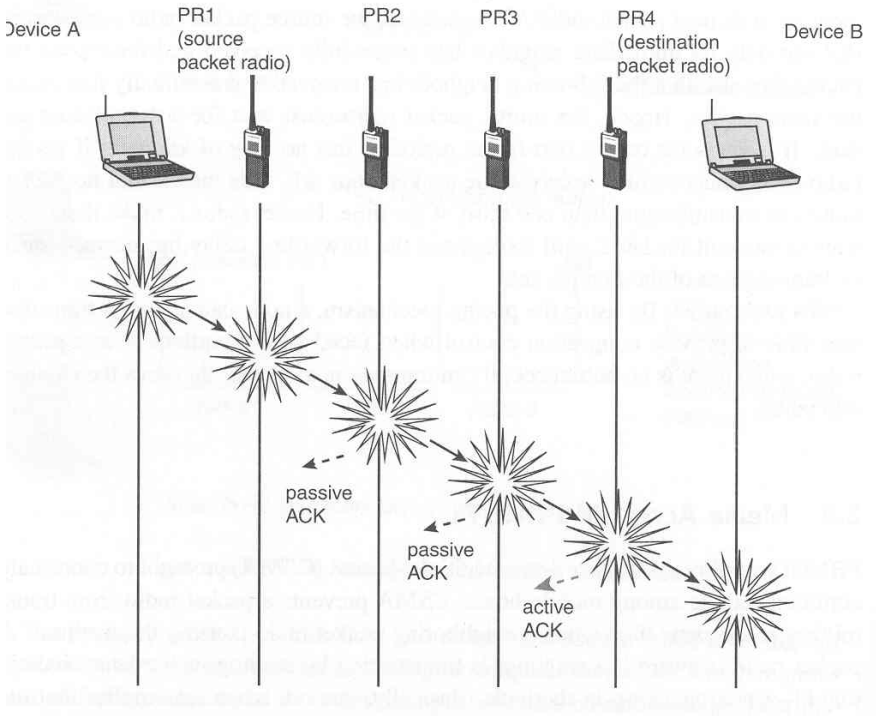

Figure 2. Passive and Active Acknowledgements in PRNET (From [2]).

C. AD HOC NETWORK CHALLENGES

Ad hoc wireless networks are formed by a collection of two or more devices that have the capability of wireless communication and networking. The ad hoc network in itself is self organizing and adaptive. Thereby these networks can be mobile, work in a stand alone mode or be networked with more than two elements. In addition there may be concurrent movement by the nodes. This will require rules to ensure consistency when multiple route reconfiguration and repair is performed.

1. Spectrum Allocation

Ad hoc networks are based on the ISM band and are required to operate over some form of allowed frequency range. The spectrum is not only tightly controlled and allocated but may be required to be purchased in the future.

2. Media Access

Cellular networks have a centralized control and a global synchronization. There are no such controls in an ad hoc network. Therefore the schemes that are used in a cellular network (TDMA, FDMA) are not suitable in ad hoc networks. Other media access control MAC protocols do not deal with host mobility. In ad hoc networks we have to deal with multiple mobile ad hoc nodes or hosts that have access to the common channel and must share the channel without a central coordination mechanism in a distributed fashion. Therefore the MAC protocol that is designed for ad hoc network must contend with problems such as collision with neighboring nodes, hidden terminal problems and exposed node problems. These problems and the mechanism to overcome them are discussed in the chapter 3 of the thesis.

3. Routing

The classical distributed Bellman-Ford routing algorithm is used to maintain and update routing information in a packet radio network. Distance-vector based algorithms could also be used in a packet radio network because the rate of mobility in this network is not high. However, these protocols will not support a truly ad hoc network as the presence of mobility implies that links make and break in a nondeterministic fashion. Therefore the current distance-vector and link state based routing protocols do not have the capability to address frequent link changes in ad hoc networks resulting in improper route convergence and inefficient communication throughput. Therefore new routing algorithms are needed. We will take a look at a few routing algorithms and list their advantages and disadvantages in chapter 4 of the thesis.

4. Energy Efficiency

All network protocols do not consider power requirements as they are meant for static hosts and routers which are powered by mains. However, in the ad hoc scenario all the devices are mobile and are generally battery-powered. These mobile devices have a superior micro processor technology and are able to process and compute enormous information. However, the battery technology has not evolved sufficiently to provide unlimited power for these devices. Therefore energy/power consumption is a major design consideration in an ad hoc network. In particular, nodes in an ad hoc mobile network require performing both the role of an end user system (user interacting system) and an intermediary system (packet forwarding system). Forwarding packets on behalf of others consumes power and thereby routing protocols have to consider energy efficiency under such circumstances. These requirements and other communication requirements are discussed in chapter 5 of the thesis.

5. Security and Privacy

Ad hoc networks are generally isolated from the internet unless they are connected to the internet through a wired node. Therefore the attacks possible are limited to attackers in the physically proximity/ radio range of the network. However, the attacker does not require any physical connectivity to the network to perform any malicious acts and this makes spoofing, passive monitoring, active jamming, interception/relaying of packets, session hijacking etc easier than in a wired environment. This requires a good neighbor node and user authentication mechanism, sound encryption and a sound session management system.

6. Congestion Control Mechanism

TCP is a connection oriented protocol most widely used to provide for flow and congestion in a wired network. This protocol relies on the round trip time of a packet to conclude if congestion has occurred. In an ad hoc network such a mechanism will be unable to distinguish network congestion as packets may be lost or dropped because of the nature of the wireless medium. Therefore certain suitable modifications or

enhancements have to be made to the existing TCP protocol so as to enable TCP support effective end to end communication throughput in a wireless environment.

III. MEDIA ACCESS CONTROL

A. INTRODUCTION

Wireless media can be accessed by any node in a shared environment. This could lead to possible contention over this common channel. In an ad-hoc network the problems are compounded as the nodes are mobile and there is no central controller. We will take a look at the existing protocols that allow efficient use of a shared wireless media. The first protocol is the MAC protocol which is concerned with a single link communication and not an end to end communication. There are two types of MAC protocols synchronous and asynchronous. Synchronous MAC protocol requires all nodes be synchronized in time and this is achieved by synchronizing to a master beacon. Therefore this requires central coordination. Asynchronous MAC protocol does not necessarily follow the same time as are contention based.

B. HIDDEN TERMINAL PROBLEM

Well known protocols such as ALOHA, slotted ALOHA, CSMA, IEEE 802.11 have a common problem known as the hidden terminal problem. When two nodes are hidden from each other (beyond radio range) they can start transmitting simultaneously thereby resulting in a collision of data at the receiver node. To avoid collision a process of hand shaking with the help of control signals such as RTS (request to send) and CTS (clear to send) were used and a new protocol developed. This protocol required nodes not to access the medium when an RTS was sent and a CTS was generated to acknowledge the RTS. The RTS and CTS signals consisted of a field that showed the time duration for which the channel is to be considered busy. This method also had its problems known as the hidden terminal problem. The hidden terminal problem occurs when RTS and CTS control messages are sent by different nodes. For example, node B is granting a CTS signal to a node A which collides with the RTS signal being sent by node D to another node C. Node D is the hidden terminal from node B and it does not receive the expected CTS from node C. This causes a retransmission of the RTS by node D for node C. Node A would have received the CTS from node B and is not aware of any collision that exists

at node C and hence proceeds with the transmission of data to node B. Under these circumstances the data would collide with the CTS being sent by node C in response to the RTS sent by node D.

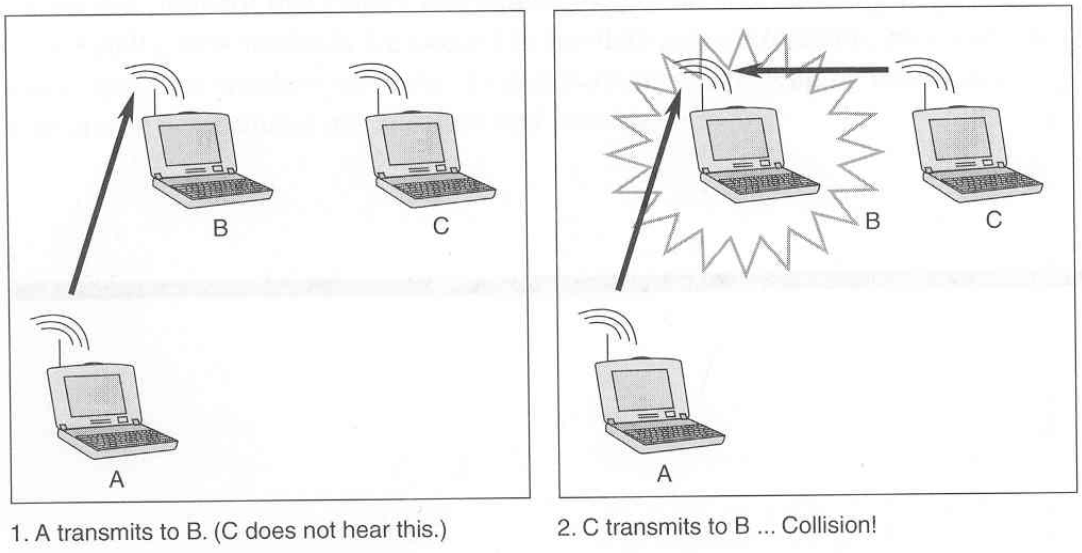

Figure 3. Hidden Terminal Problem. (include reference to where this picture is taken and do so for all such pictures) (From [2]).

Another problem could occur when multiple CTS messages are granted to different neighboring nodes leading to collisions. For example, when node B returns a CTS message in response to an RTS message sent by node A and node C sends an RTS message at the same time to node B. Node C is unaware of the communication between node A and B and also sends an RTS to node D. Node D proceeds to grant CTS message to node C. Under this scenario both nodes A and C start transmitting data leading to a collision.

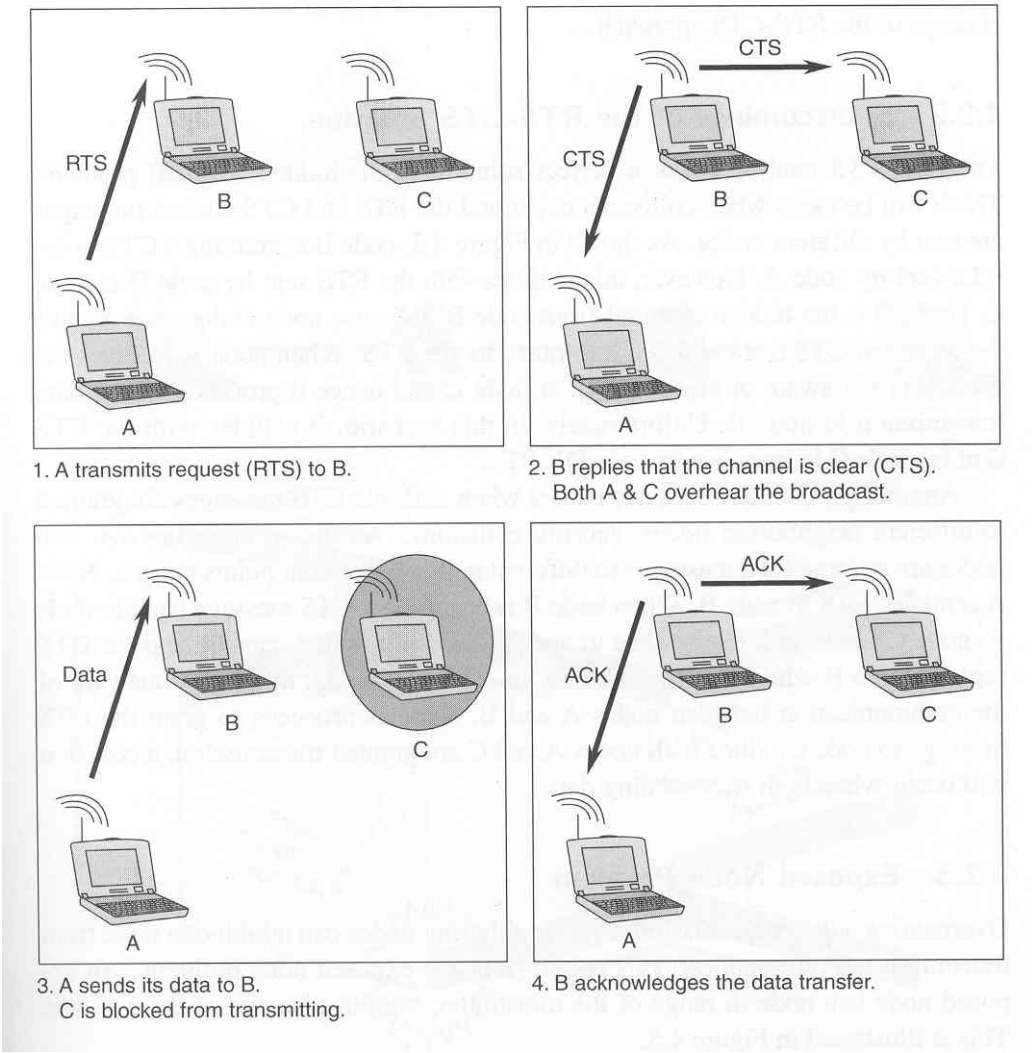

Figure 4. Hidden Node Problem Resolution by RTS-CTS Handshake (From [2]).

C. EXPOSED NODE PROBLEM

Data transmissions occurring from neighboring nodes will inhibit one node from transmitting to other nodes as the node overhears the transmission. This is called an exposed node problem. A node in range of the transmitter but out of range of the receiver is called an exposed node. Under such conditions the exposed node tries communicating without success and also inhibits other nodes from communicating successfully. The use of a separate control and data channel or the use of directional antenna would eliminate this problem.

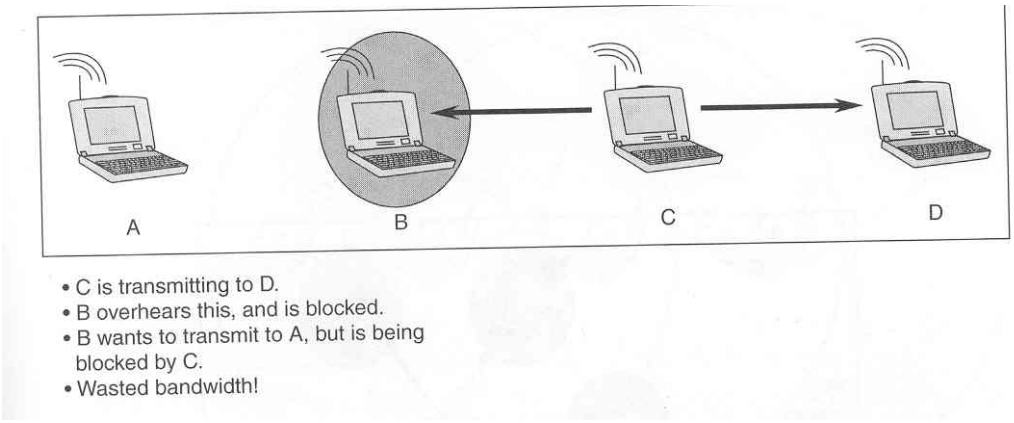

Figure 5. Exposed Node Problem (From [2]).

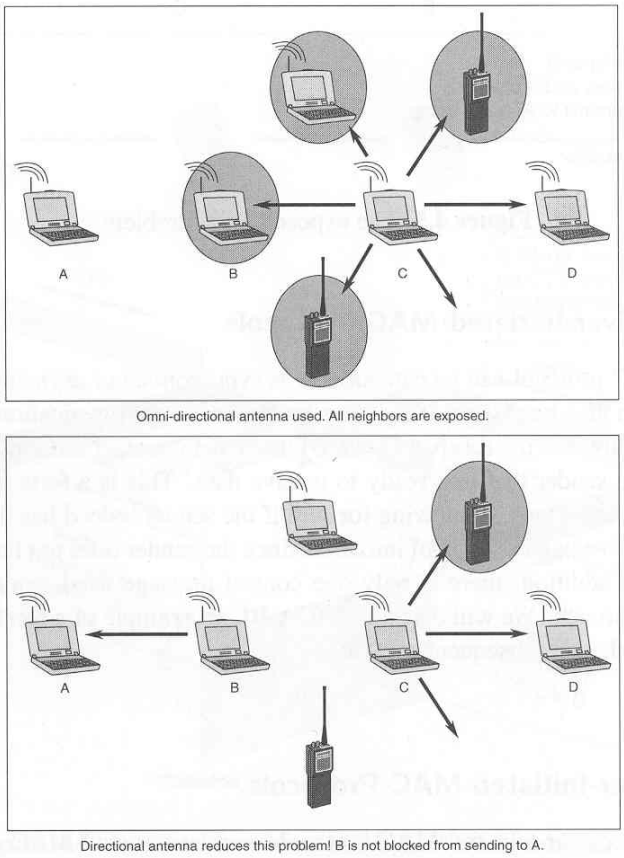

Figure 6. Solution of Exposed Node Problem by Unidirectional Antennae (From [2]).

D. CSMA-CD

The multiple access collision avoidance (CSMA-CA) technique is dependant on a three way hand shake, RTS-DTS-DATA. The main feature of this protocol is that it inhibits a transmitter when a CTS packet is overheard. Thereby collision does not occur even during the RTS-CTS phase. The mobile hosts add a random amount of time to the minimum interval required waiting after an RTS or CTS control message is sent. If two or more stations transmit an RTS concurrently resulting in a collision, these stations will wait for a randomly chosen interval before trying again. This interval is doubled if there is a collision on every attempt thereafter. The station that succeeds will receive CTS from its responder and thereby block other stations from transmitting to allow the data communication session to begin.

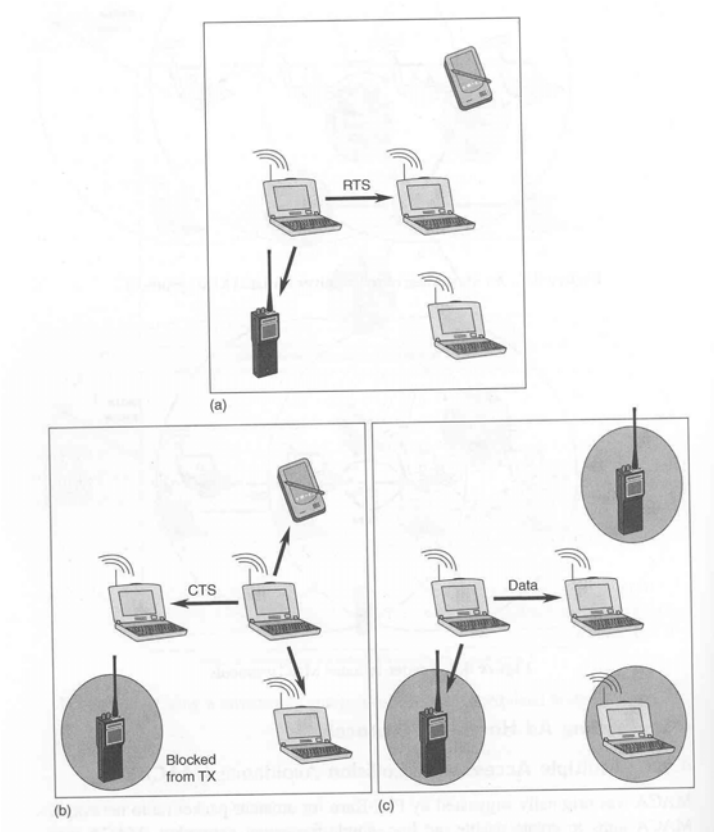

Figure 7. Controlled Handshake used in CSMA-CA (From [2]).

E. CSMA-CD-BI

There is another multiple access collision avoidance mechanism (CSMA-CA-BI) which uses only a two way handshake. In this case there is no RTS instead the message is reconfigured as RTR (ready to receive). This mechanism is called multiple access collision avoidance by invitation. The RTR is sent when the receiver node is ready to accept data. In this scenario the receiver does not necessarily know that the sender has data to transmit. The communication performance in this protocol is dependant on the timely invitation sent by the receiver. This time period is estimated based on the packet queue length and arrival rate at the source. This is accomplished by sending the packet queue length and arrival rate information along with each data packet so as to enable the receiver is aware of the transmitter's backlog. This scheme is successful when the data rate is constant and the performance drops when the traffic comes in bursts. In the worst case scenario this mechanism will perform no better than the multiple access collision avoidance case.

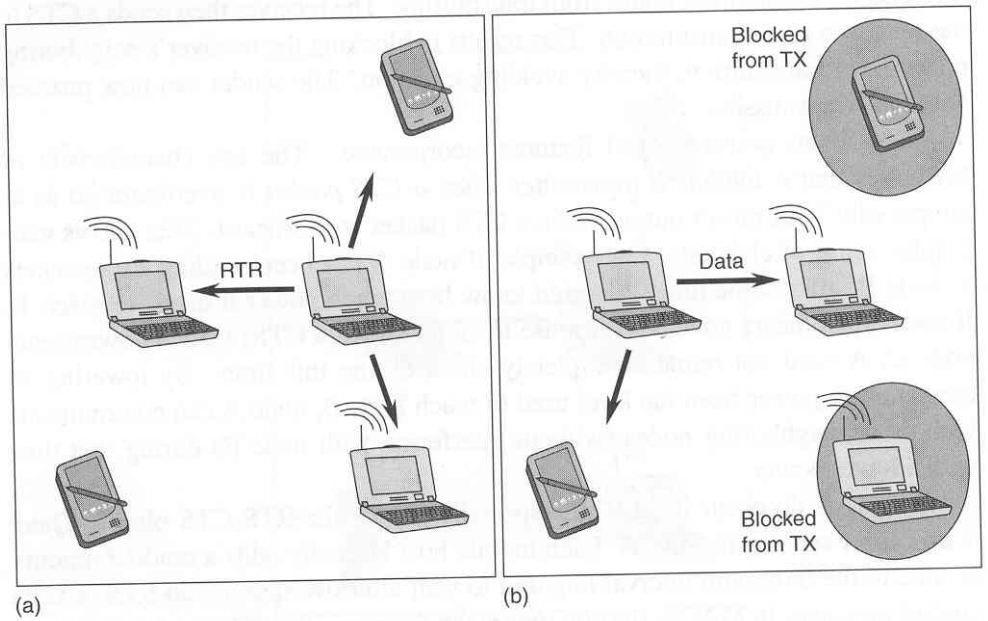

Figure 8. CSMA-CA BI Handshake (From [2]).

F. PAMAS

The power aware multiple access protocol with signaling (PAMAS) is used in ad hoc networks and is based on the CSMA-CA with the additional signaling channel. This protocol requires the conservation of battery power by selectively powering off non active nodes or nodes not receiving any package. Conditions that force a node to power off mode are:

i. Node has no packets to transmit and one of its neighbors is transmitting.

ii. A node has packets to transmit but at least one of its neighboring nodes is transmitting or is receiving.

Suppose a node A wishes to transmit, it sends an RTS and enters a wait-for-CTS state. If the CTS does not arrive the node takes an exponential back off state and retries to send the RTS later. If the CTS arrives node A enters transmit data state. The receiver node B sends a CTS and enters the await data state. If the data arrives the node will transmit a busy tone over the signaling channel and enter receive data state.

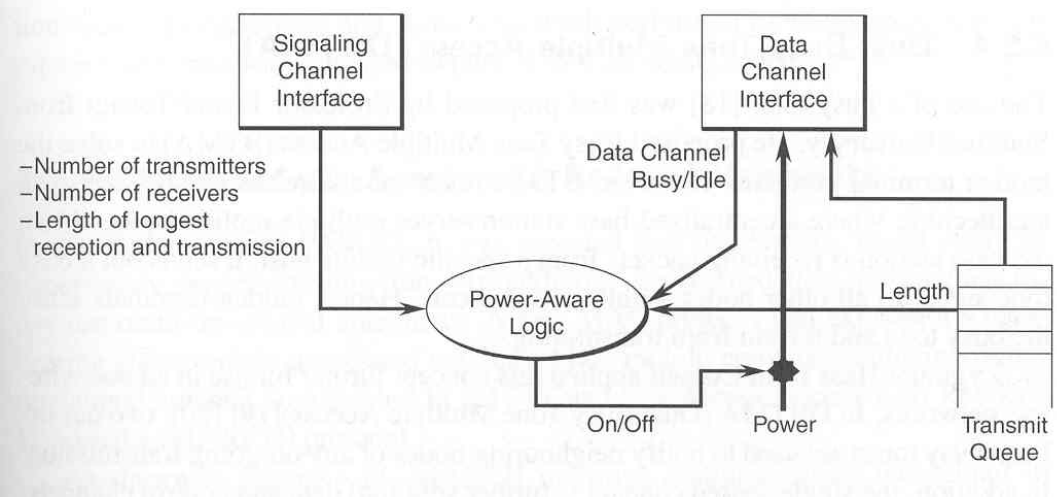

Figure 9. PAMAS MAC interfaces and Power Aware Logic (From [2]).

G. DBTMA

DBTMA (dual busy tone multiple access) uses two out of band busy tones and a single shared channel split into data and control channels. Data packets are sent over the data channel and the control signals over the same shared channel. When a station wants

to transmit it first sends out an RTS message. The receiver node willing to accept the data sends out a receive-busy tone followed by CTS message. Upon receiving this CTS message, the source transmits a busy tone message prior to data transmission. Neighboring nodes that hear transmit busy refrain from transmission and discard any received packets.

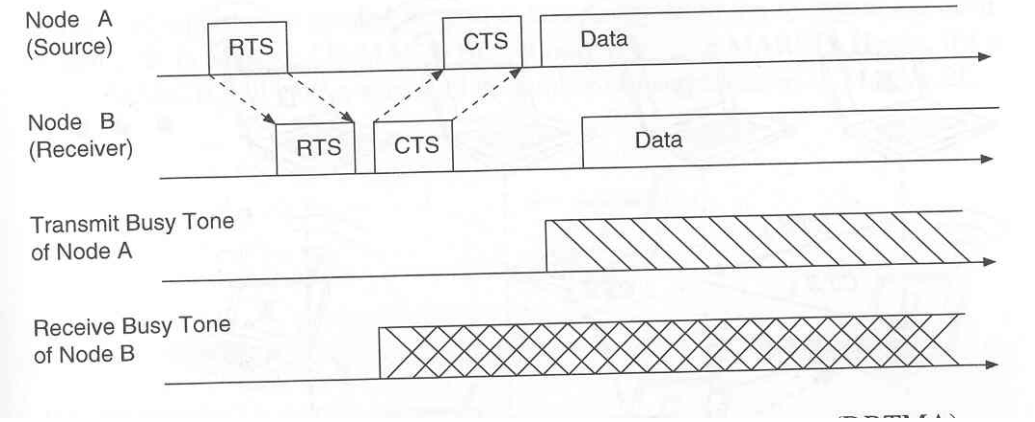

Figure 10. DBTMA Principle (From [2]).

H. MARCH

MARCH (media access with reduced handshake) improves the transmission of packets by reducing the amount of control overhead. It uses the characteristics of omni directional antenna to reduce the required number of handshake signals. Adjacent nodes B and C in a route will be able to monitor the CTS signal transmitted by the node B to its source node A. Thereby node C can send CTS to node B without a RTS and the data can be forwarded from the node B to C for onward transmission. This is illustrated in the figure below. This protocol is a request first pull later protocol as the subsequent nodes in the path just need to send invitation to pull data towards them.

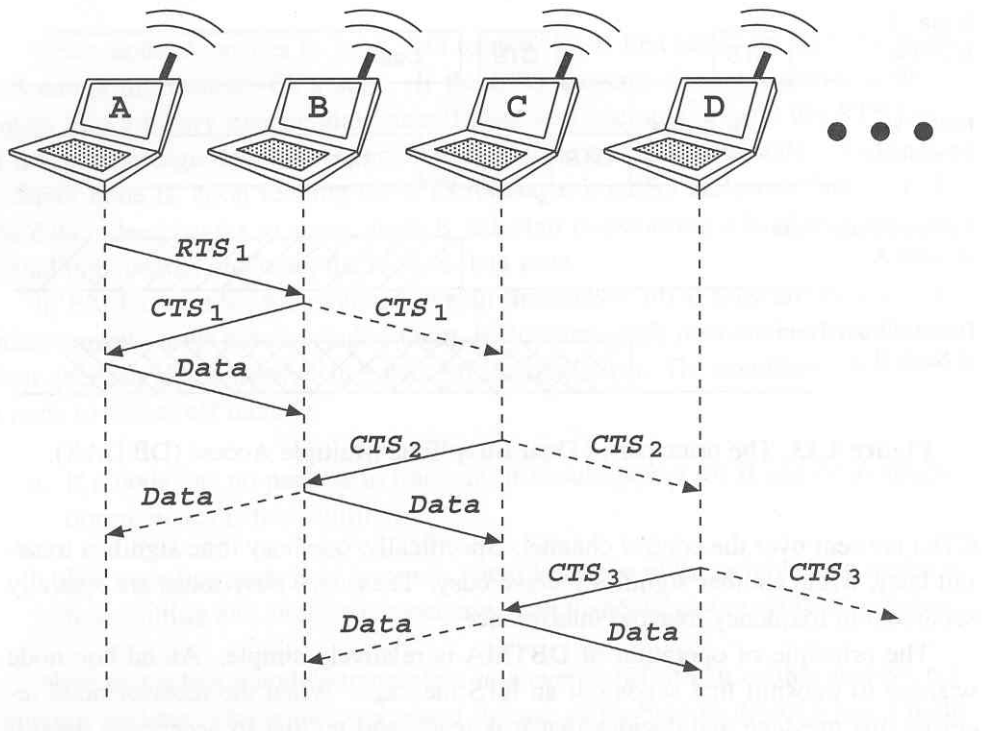

Figure 11. Handshake mechanism of MARCH (From [2]).

I. IEEE 802.11 STANDARD

IEEE 802.11 standard supports a peer to peer (IBSS) independent basic service set which is an ad hoc network with all its station within each others transmission range. Hence this technology becomes the obvious choice when setting up an ad hoc network. Let us take a look at the evolution of this standard. IEEE, European Telecommunication Standards Institute and One Technology Alliance (Home RF) promote wireless local area network standards (WLAN). Among the standards that were instituted by these organizations the leading standard was the IEEE 802.11 standard. Among the IEEE 802.11 standards, 802.11b (Wi-Fi) was the leader. 802.11a and 802.11g were the challengers to this standard. 802.11b is based on the (DSSS) direct sequence spread spectrum version of 802.11 using a 2.4 GHz spectrum and orthogonal frequency division multiplexing (OFDM). Its counterpart 802.11a uses the 5GHz spectrum. The 2.4GHz spectrum is capable of penetrating physical barriers such as walls and ceilings and is unlicensed in most of the countries. 802.11b supports a raw data transfer rate of 11Mbps up to a distance of 30meters. The rate drops to 5.5Mbps beyond 30 meters and drops to

2Mbps around 65meters. The maximum range is 100meters and the data rate is 1Mbps. The interesting point about the 802.11b specification is that it supports roaming between access points. The specification requires a method for roaming but leaves the implementation to the manufacturer. A feature known as the wired equivalent privacy (WEP) was created with the 802.11 specification to provide security. The intention was to provide a basic level of authentication and privacy by the use of data encryption. For authentication an access point will request a client to verify its identity by sending a text. The client will use RC4 encryption with a secret key to encrypt the text and return it back to the access point. The access point will then decrypt the text using the same key and verify whether the client is to be granted access or not. The WEP consists of a 24bit initialization vector in addition to the WEP key. This vector is changed after a certain interval. However, this initialization vector is sent in the clear and thereby the WEP security is not very strong. Additional security mechanisms are being incorporated into this standard by a task group and the new standard draft is currently being named as the 802.11i standard.

Synchronization is necessary for frequency hopping spread spectrum (FHSS) to ensure that all stations “hop” at the same time: it is also necessary for both FHSS and DSSS to perform power management. In an IBSS that is not very small there is a non negligible probability that the stations may get out of synchronization. As the number of stations increase, the probability of asynchronism also increases. To alleviate this problem modifications have been suggested to the current synchronization algorithm of the 802.11 standard. Therefore the IEEE 802.11 IBSS is a fully connected single hop ad hoc network with all station within each others range. Investigations are currently being done to see if the 802.11 standard can support multi hop ad hoc networks assuming each station has the capability of routing.

IV. AD HOC ROUTING PROTOCOLS

A OVERVIEW

Since the advent of packet radio networks in the early 70's by DARPA numerous protocols have been developed for ad hoc mobile networks. Many of these protocols require dealing with power consumption, low bandwidth and high error rates. These protocols can be characterized as:

i) Table driven

ii) Source initiated on demand driven

The table driven approach attempts to maintain consistent, up to date information for each node to every other node in the network using tables. They respond to changes in the network by updating the required information and also propagating route updates to maintain a consistent view. The source initiated on demand approach creates routes only when desired by a source node. When the node requires a route to a destination a discovery process is initiated and is completed once a route is found. The process examines all possible permutations before suggesting a route. Once the route is discovered it is maintained by a route maintenance procedure. If the route becomes inaccessible or is no longer desired, it is dropped by the maintenance procedure.

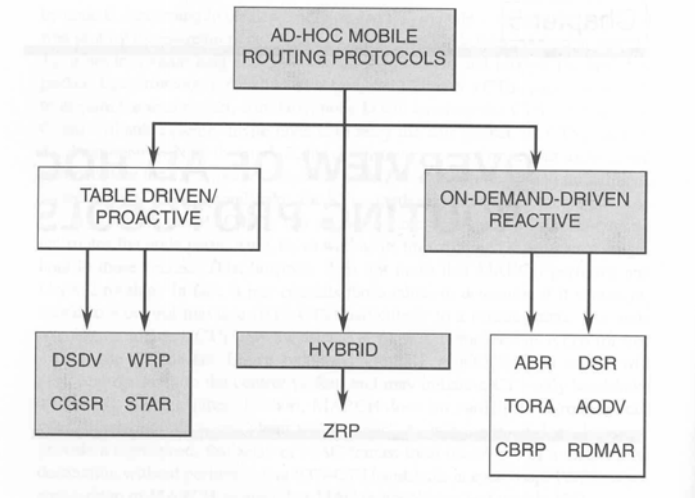

Figure 12. Ad hoc Routing Protocols (From [2])

B TABLE DRIVEN APPROACHES

The table driven approaches are –

a) Destination Sequenced Distance Vector (DSDV)

b) Wireless Routing Protocol (WRP)

c) Cluster Switch Gateway Routing (CSGR)

1. Destination Sequenced Distance Vector (DSDV)

This protocol is based on the classical distributed Bellman-Ford algorithm. However, loops are avoided in this algorithm by the use of routers. Every node maintains the route information in a table of all the possible destinations within a non-partition network and the number of hops to each destination if required. Therefore the route is always available regardless of whether it is required or not. Routes are distinguished by providing a sequence number. Updates are done periodically through out the network to maintain consistency and this rendering causes a lot of control traffic which reduces the utilization of network resources. To alleviate this problem route updates are performed using one of the two route update packets. The first known as full dump is performed infrequently when there is less traffic and this packet consists of all available routing information. This information is sent in multiple network protocol data units. The second known as incremental packets are used to relay only changes since the last full dump. Broadcasts containing new route information will consist of the address of the destination, the number of hops to reach the destination, a sequence number of the information received regarding the destination as well as a new sequence number unique to the broadcast. The route labeled with the most recent sequence number (in increasing order) is always used and the route with the smaller hop count is used in case the sequence number is the same for more than one route.

2. Wireless Routing Protocol (WRP)

Wireless routing protocol belongs to a family of path finding algorithms which avoid loops. In this algorithm the nodes communicate information regarding the distance and the second-to-last hop in the network. This algorithm avoids the count to infinity

problem caused by loops by forcing every node in the network to perform consistency checks of predecessor information reported by all its neighbors. This process avoids looping situations and also provides an efficient route convergence when a link failure happens. The existence of nodes is informed by the neighbors acknowledging the receipt of transmitted requests and other messages. When a node is not sending any packets it must send a hello message within a specific period of time that informs that the node is properly connected. If there is a lack of message from a particular node then it indicates a link failure. When a hello message is received that new node information is added to the routing table and a copy of this table is sent to its neighbors.

WRP maintains four tables viz., a) Distance table b) Routing table c) Link cost table and d) Message retransmission list. The number of hops between a node and its destination is stored in the distance table. The next hop information is stored in the routing table. The delay information associated is stored in the link cost table. Sequence number of the update message, a retransmission counter, an acknowledgement required flag vector and a list of update messages sent are stored in the message retransmission list. Retransmitted messages must be acknowledged by the neighbors and such information is stored as updates in the update message of the MRL.

3. Cluster Switch Gateway Routing (CSGR)

In this form of routing nodes are grouped into clusters with every cluster having a head. Such a grouping introduces a form of hierarchy. The cluster head controls the group and this framework provides for code separation, channel access, routing and bandwidth allocation. The cluster head is chosen using a distributed cluster head selection algorithm. When a cluster head leaves the group a new head is selected. This could lead to problems if the cluster head changes frequently as the group will then spend time choosing a new cluster head instead of forwarding data towards their intended destinations. To avoid this phenomenon a least cluster change (LCC) algorithm is introduced. This algorithm effects a change only when two cluster heads come into contact or when a node moves out of all other cluster heads. CSGR basically employs the DSDV routing scheme with a slight modification. The modification involves the use of cluster-head-to-gateway routing approach to send packages from the source to its destination. Nodes that are within

communication range of two or more cluster heads are called gateway nodes. A packet sent by a node is first routed to its cluster head and then through a gateway it is routed to the cluster head of the destination node. Once the destination node cluster head is reached the packet is then transmitted to the destination. Therefore each node must keep a cluster member table. This table information is broadcast by every node using the DSDV protocol periodically. In addition a routing table is also maintained to keep track of the next hop information. When a packet arrives, the next hop information and the cluster member information of the nearest cluster head along the route is processed and the message is transmitted. Update information regarding both routing and cluster member tables is needed in the CSGR approach.

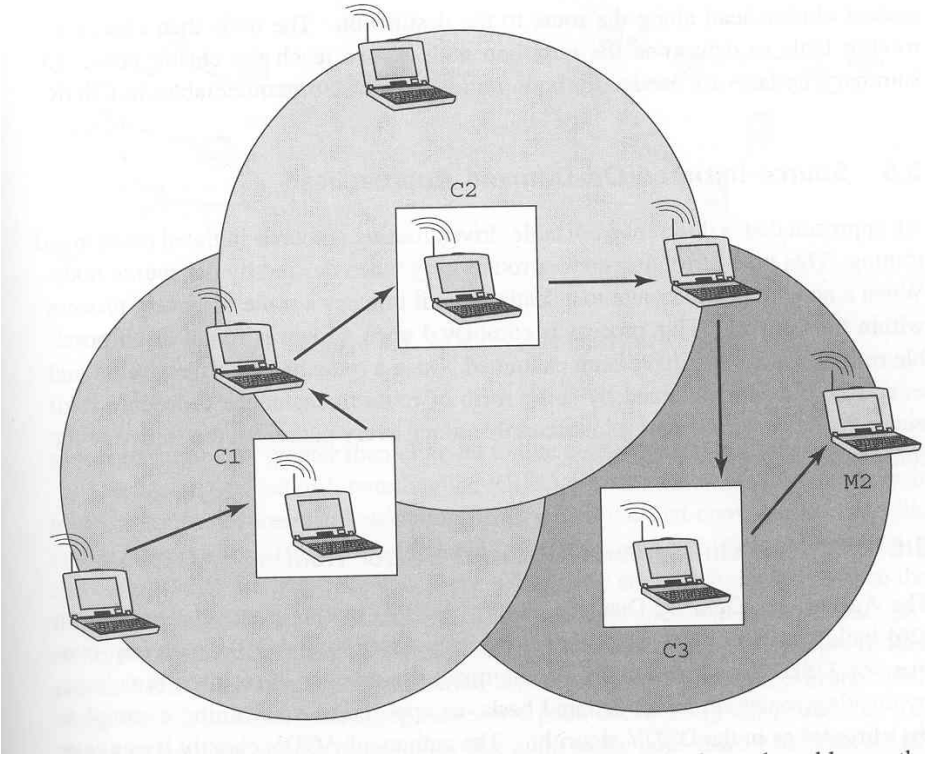

Figure 13. CSGR path constrained into cluster heads (From [2]).

C. SOURCE INITIATED ON DEMAND APPROACHES

The source initiated on demand approaches are –

a) Ad hoc On Demand Distance Vector Routing (AODV)

b) Dynamic Source Routing (DSR)

c) Temporally Ordered Routing Algorithm (TORA)

1. Ad Hoc On Demand Distance Vector Routing (AODV)

The AODV protocol is an improvement of the DSDV approach as it minimizes the number of required broadcast by generating the routes on an on demand basis as apposed to maintaining the complete list of routes. Routes that are not the selected path do not maintain any routing information and do not participate in any exchange of any routing table information. Messages sent to a destination by a source generate a path discovery process when a valid route to the destination does not exist. When a source node has to send a message it broadcasts a route request packet (RREQ) to all its neighbors which is then forwarded by the neighbors to their neighbors and so on until the destination is reached or an intermediate node with a recently updated route to the destination is located. AODV maintains destination sequence numbers and recent route information to all nodes which ensures the routes are loop free. Every node has a broadcast ID and a sequence number. The broadcast ID together with the nodes IP address uniquely identifies an RREQ which is incremented every time a path discovery process is initiated. A message containing the sequence number, broadcast id and the most recent RREQ is sent when the source node requires communicating with the destination. Intermediary nodes reply to the RREQ only if there is a route to the destination whose corresponding destination sequence number is greater than or equal to that contained in the RREQ. Intermediary nodes record in their route tables the address of the neighbors from which the first copy of broadcast packet was received when they forward the RREQ and thereby establish a reverse path. If multiple copies of the same RREQ are received they are discarded. When the destination is reached, the destination or an intermediary node responds by sending a route reply (RREP) along the reverse path. As the RREP is routed back along the reverse path intermediary nodes in this path

set up route entries in their table that point to the node from which the RREP was received. These route entries indicate the active forward path. Every route entry is associated with a timer and is deleted if the entry is not used within the specified life time. When a source node moves it has to reinitiate the discovery process to find a new route to the destination. If an intermediary node along the route moves then its upstream neighbor propagates a link failure notification message (RREP with infinite cost) to each of its active upstream neighbors and requires them to erase that part of the route information. This notification is further propagated by the upstream neighbors to their active upstream neighbors and so on. Hello messages are periodic local broadcasts made by a node to inform its presence to its neighbors may be additionally used to indicate the existence of a link. However,, the use of these messages is not required as the nodes listen for retransmissions of data packets and if such a retransmission is not heard the node assumes a link failure. It may then request acknowledgement for network connectivity using the hello messages or any other message.

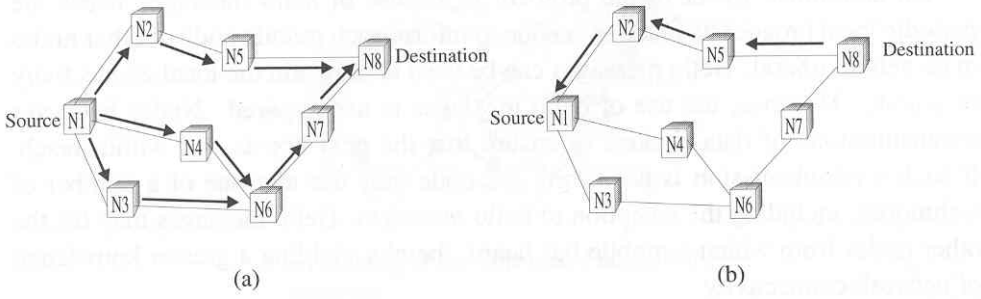

Figure 14. AODV route discovery process (From [2]).

2. Dynamic Source Routing (DSR)

Dynamic source routing is based on the concept of source routing. In this routing nodes are required to maintain route caches that contain routes which the nodes are aware of. Entries are continually updated into the route cache as new routes are discovered. The protocol consists of two major processes a) route discovery and b) route maintenance. When a packet is available at a source node for dispatch for some destination, the source node first consults its route cache to determine if there exists an unexpired route to that

destination. If such a route exists the packet is sent using that route and if it doesn't a route discovery process is initiated by broadcasting a route request packet. The route request message consists of the address of the destination along with the address of the source node and a unique ID. Each node receiving this route request message checks its cache to see if a route to the destination exists before forwarding the message along its outgoing links. To limit the outward propagation of route requests a mobile node forwards the route request only if the request has not been seen so far or the mobile's address has not appeared in the route record. A route reply is generated when the route request reaches the destination or an intermediary node which has an unexpired route to the destination on its cache. The route reply specifies the route from the source to the destination. Route maintenance is done by the use of a route error packet and acknowledgements. Route errors are generated when there is a fatal transmission problem. Route error messages are also used to verify the proper functioning of the route links. Acknowledgements to route error messages could also be passive acknowledgements.

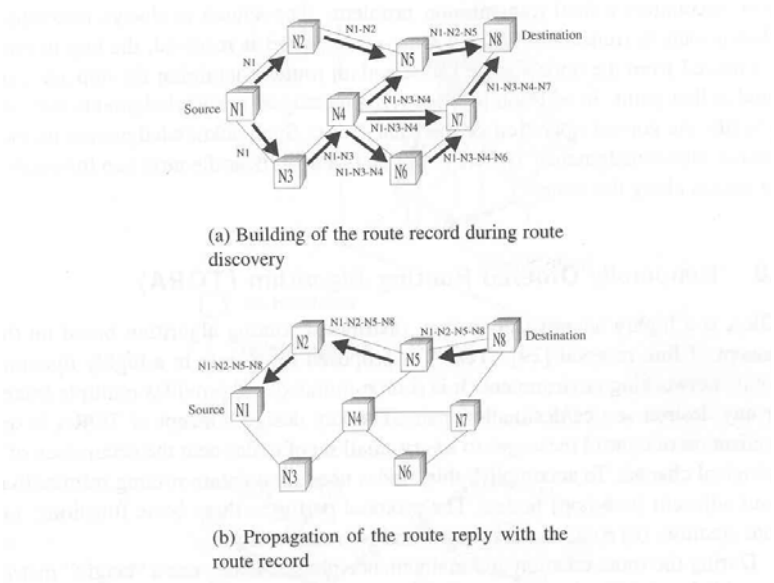

Figure 15. DSR source route creation process (From [2]).

3. Temporally Ordered Routing Algorithm (TORA)

TORA is based on the concept of link reversal which is a highly adaptive loop free distributed routing algorithm. The most essential requirement of a TORA is the

localization of control messages to a very small set of nodes when a topological change occurs to the network. This protocol is source initiated, suited for a highly dynamic mobile networking environment and provides multiple routes for any source destination pair. It performs three basic functions: a) route creation b) route maintenance and c) route erasure. The route creation and maintenance process establishes a DAG (directed acyclic graph) between the source and the destination using a “height” metric. Nodes are there after assigned upstream or downstream based on their relative “height” metric compared to their neighboring nodes. If a node moves the DAG is broken and route maintenance becomes necessary to reestablish a DAG routed at the same destination. TORA is dependant on a clock timer and assumes all nodes are synchronized to the clock. The calculation of the height metric is based on the timing. This metric comprises of five elements namely a) logical time of link failure b) unique ID of the node that defined the new reference level c) reflection indicator bit d) propagation ordering parameter and e) unique ID of the node. The first three elements collectively represent the reference level. Every time a node looses its last link due to link failure a new reference level is defined. This scheme has a potential for oscillations to occur. However, such oscillations are temporary and route convergence does occur finally as TORA uses inter nodal coordination.

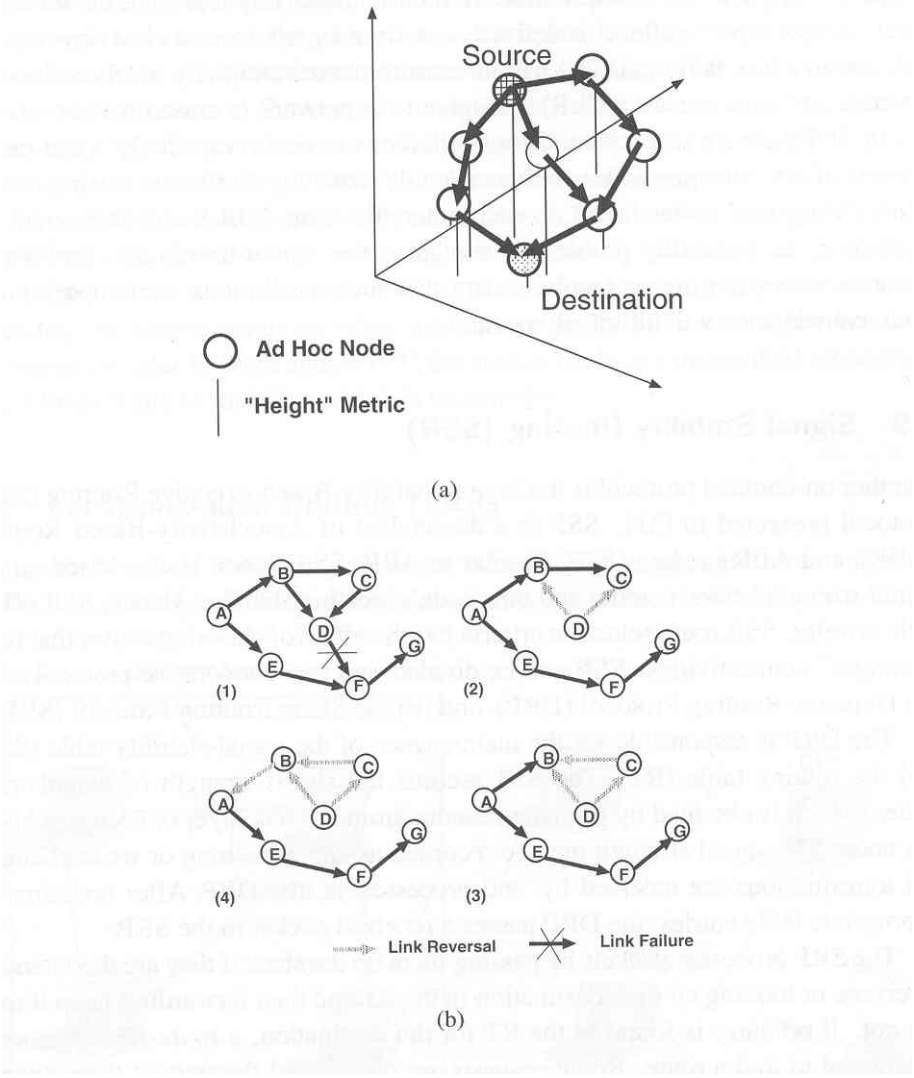

Figure 16. (a) Route creation (b) Route maintenance (From [2])

4. Signal Stability Routing (SSR)

SSR performs route selection based on the signal strength between nodes and on a node's location stability. SSR can be divided into a) dynamic routing protocol (DRP) and b) static routing protocol (SRP). A signal stability table (SST) and a routing table (RT) is maintained by the DRP. The SST contains records of signal strength of neighboring nodes which are obtained periodically from beacons sent out by the neighbors at the link layer. The signal strength is categorized as either strong or weak channel. The routes from source to destination are stored in the routing table. When a packet is received for transmission it is processed in the SRP. The SRP looks for the destination of the packet in

the RT and then forwards them if a route is found. If no route is found then a route search process is initiated and route requests propagated through the network. However, the route requests are forwarded to the next hop only if they are received over strong channel and have not been processed previously. The destination returns the route search packet that it received first as it is most probably the packet that has traveled the shortest path or the least congested path. The DRP then reverses the selected route and sends a route reply message to the source. The source then forwards the message to the destination along the route specified in the route reply message. Route search packets that are received at the destination are assumed to have traveled the path of the strongest signal links as the nodes would have been dropped if they had arrived over a weak channel. However, if no route message is received at the source within a specific time out period the source may change the preference field of the header in the route search packets to indicate the acceptance of weak channels also. Link failure is detected within the network when intermediary nodes send error messages to the source indicating the failure of a channel. Under these circumstances another route search process may have to be initiated. There after the source sends a message to erase the route to all concerned nodes.

5. Location Aided Routing (LAR)

LAR uses location information (via a GPS) to improve performance. LAR uses a) expected zone and b) request zone to limit the search for a new route. LAR assumes the sender has the knowledge of the destination and velocity before hand. The expected zone is calculated based on this information. The smallest triangle that includes the expected zone and the sender is called the request zone. Nodes that do not fall within the request zone discard packets that they receive. Nodes that fall within the request zone respond with routes to the destination if they have any. LAR depends on GPS receivers being able to be connected to the sender and other nodes. However, not all devices can be equipped with GPS receivers and in such cases the scheme will not be implement able. This scheme has to overcome problems caused due to positional errors.

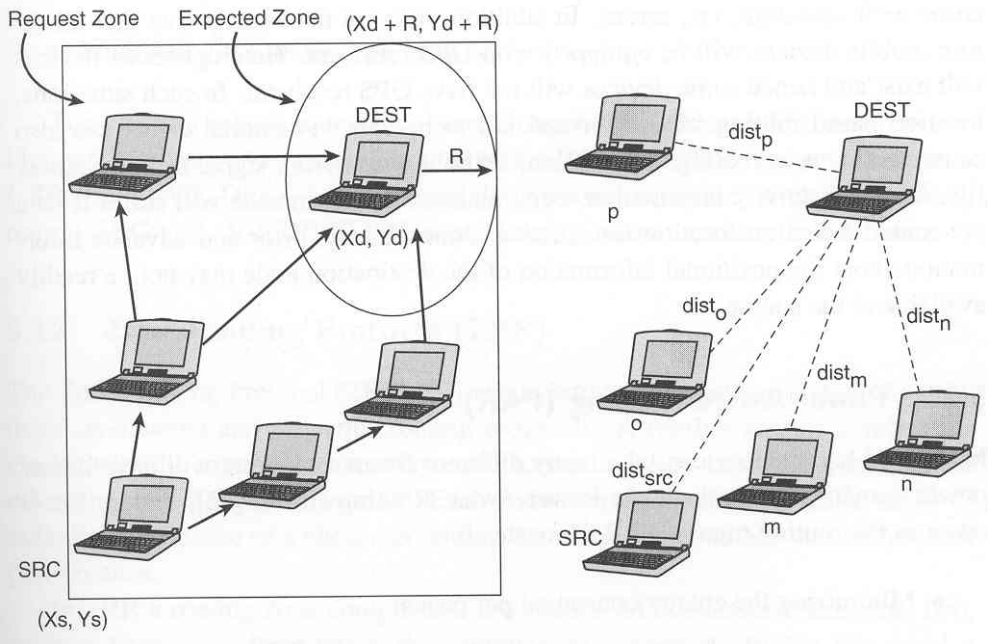

Figure 17. (a)Concept of request zone and expected zone in LAR (b) Consideration of route physical distance (From [2]).

6. Power Aware Routing (PAR)

PAR advocates:

1. Minimizing energy consumed per packet
2. Minimizing variance in node power
3. Minimizing cost per packet
4. Minimizing the maximum node cost
5. Maximizing the time before network is partitioned

This protocol prefers routes of nodes that have longer battery life.

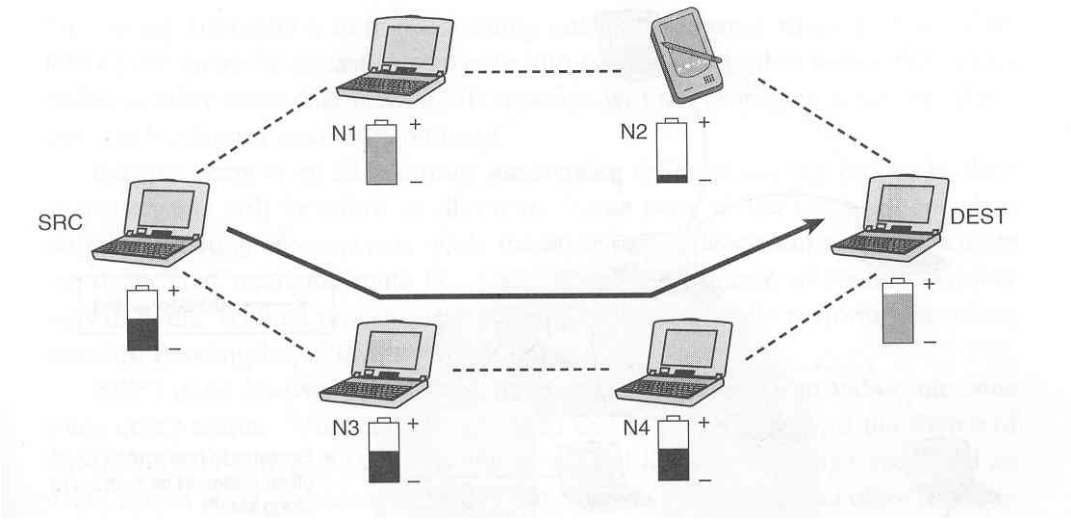

Figure 18. PAR based on remaining battery life (From [2]).

7. Zone Routing Protocol (ZRP)

ZRP is a hybrid protocol that incorporates the principles of on demand and proactive routing protocols. It consists of routing zones similar to a cluster and every node in the cluster acting as both a cluster head and a member of the cluster. Zones comprise of a few mobile ad hoc nodes with one or two hops and can overlap. A table driven routing protocol is used within the zone. It has three sub protocols a) proactive (table driven) intra zone routing protocol (IARP) b) reactive inter zone routing protocol (IERP) and c) border cast resolution protocol (BRP). IARP's role is to ensure that all nodes within the zone have a consisting routing table and can be implemented using the existing link state or distance vector routing protocols. IERP runs on the border nodes and requires them to perform on demand routing to search for routing information to nodes outside the current zone. Broadcasts are not allowed to penetrate into nodes within other zones. When routes are broken if the source for the breakage is within the zone it is treated like a change event and an event driven route update informs the other nodes within the zone of the link failure. If the mobility is a result of a border node or a node outside the zone then a route search is to be performed again.

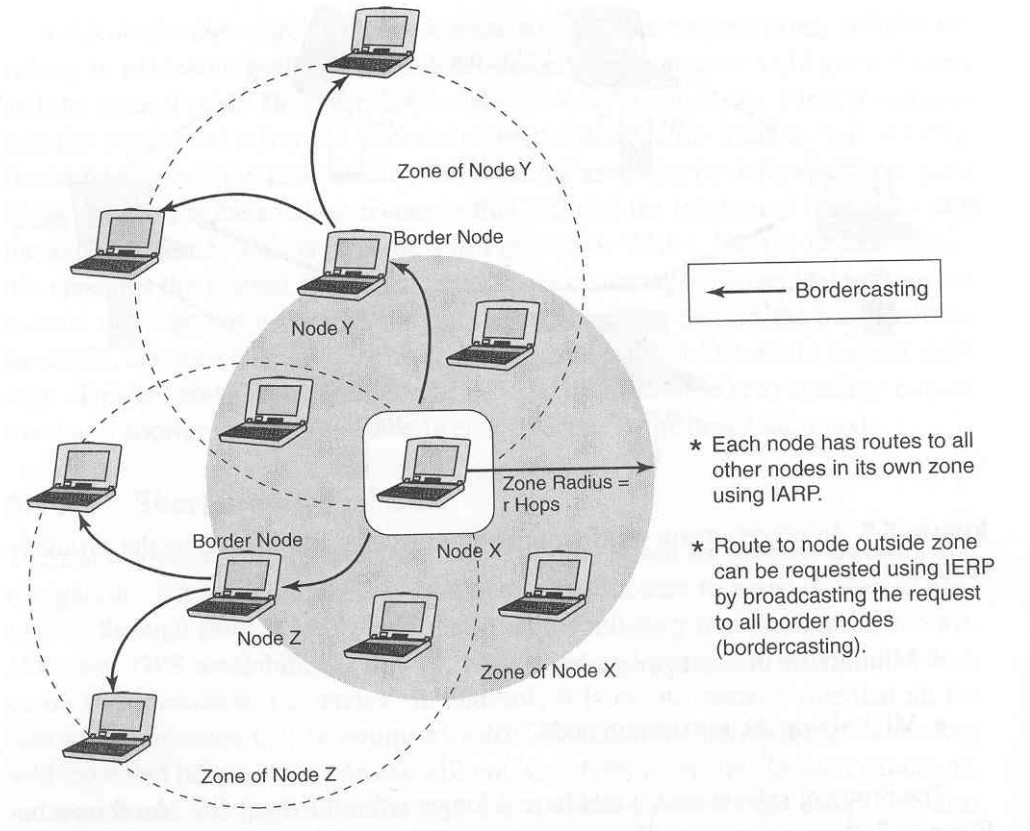

Figure 19. Hybrid approach – ZRP (From [2]).

THIS PAGE INTENTIONALLY LEFT BLANK

V. AD HOC NETWORK PERFORMANCE REQUIREMENTS

A. INTRODUCTION

Communication networks are required to transport user traffic from source to destination. The performance of such network affects the satisfaction level of users. A higher level of satisfaction is achieved when the network is able to transmit the message within a short duration of time and at a low cost. The dynamics of the ad hoc network and its underlying protocols must be able to support the transfer of information without delay. In this chapter we will look at the various performance parameters of interest.

B. SIGNIFICANT PERFORMANCE PARAMETERS

The significant parameters that emphasize the communication requirements of an ad hoc wireless network are –

a) Route Discovery Time

b) End to End Delay Performance (EED)

c) Communication Throughput

d) Packet Loss performance

e) Route reconfiguration

1. Route Discovery Time

Ad hoc protocols that are source initiated on demand based initiate a route discovery process whenever a packet is to be sent from the source to the destination. The time required for the process to discover such a route is known as the route discovery time. This process involves the sending of a route request packet by the source node which is broadcast to all its neighbors in search of valid routes to the destination. A reply control packet is then received by the source which was sent by the destination via the reverse path. The route discovery time has a direct relation with the beaconing interval in an ad hoc network. Periodic beaconing is implemented in most protocols of the ad hoc network to gather longevity information. These beacons are control messages that are

broadcast by a node to its immediate neighbors. The beaconing interval is important as it has strong implication on a) the amount of bandwidth and power consumed b) the accuracy of longevity information and c) time to detect and repair link failure. The route discovery time increases with the increase in the frequency of beaconing. The route delay increases with the increase in route length because of the increase in total propagation delay.

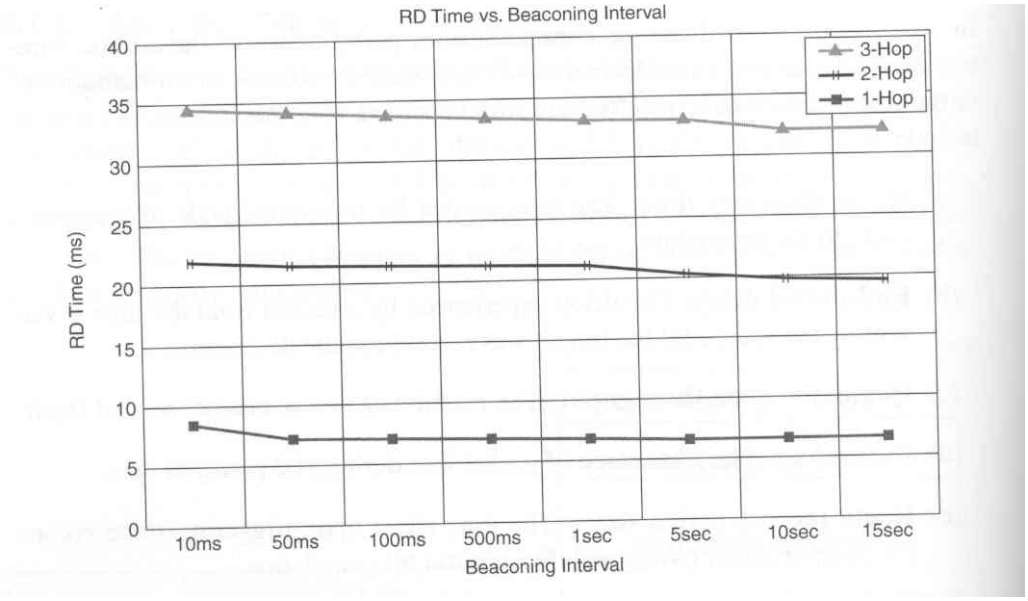

Figure 20. Route discovery time vs. beaconing interval (From [2]).

Route Length	Average RD Time
One Hop	6.64 ms
Two Hops	20.94 ms
Three Hops	32.98 ms

Table 4. Average RD Time at one beaconing interval for different hop counts (From [2])

2. End to End Delay (EED) Performance

The EED accounts for all the delays from the source to the destination. The various delays are transmission delay, propagation delay, processing delay and queuing delay. The EED various with the packet size of the message. This is because the larger the packet size the longer the elements of transmission propagation and processing times. The EED also increases with the increase in hop counts. The beaconing interval does not have any significant bearing on the interval of EED.

Average EED			
Hop Count	**Min Size**	**1000 bytes**	**Max Size**
One Hop	3.25 ms	10.40 ms	340.00 ms
Two Hops	6.20 ms	19.70 ms	26.20 ms
Three Hops	8.80 ms	29.20 ms	38.30 ms

Table 5. Average EED at minimum, 1000 bytes, maximum packet size for different hops (From [2]).

3. Communication Throughput

Communication throughput is measured by sending a *ping* message from the source to the destination and measuring the ratio between the total data received and the total delay incurred. The impact of packet size on throughput shows that there is a dramatic increase in throughput with an increase in packet size beyond 64bytes. However, the throughput decreases when the packet size reaches 128bytes and stagnates until it reaches a saturation point. The beaconing interval does not have an impact on the throughput except at extremely high beaconing frequency. The increase in route length causes degradation in the throughput.

4. Packet Loss Performance

Packet loss is an important measure in an ad hoc network as the network is subject to external interference and multi path fading which affect the end to end communication. To measure such packet loss 100 *ping* messages were sent and the amount of packet lost in transmission recorded. At higher beacon frequencies packet losses tended to be higher and also as the packet size increased there were significantly higher packet losses. Increasing the beaconing interval did not have any significant effect on packet loss. The route length also did not have any significant influence on packet loss. It was However, observed that collusions in a channel could significantly contribute to packet loss. Therefore in an ad hoc route lengthened by multiple hops the overall path loss is dependant on intermediary links along the route.

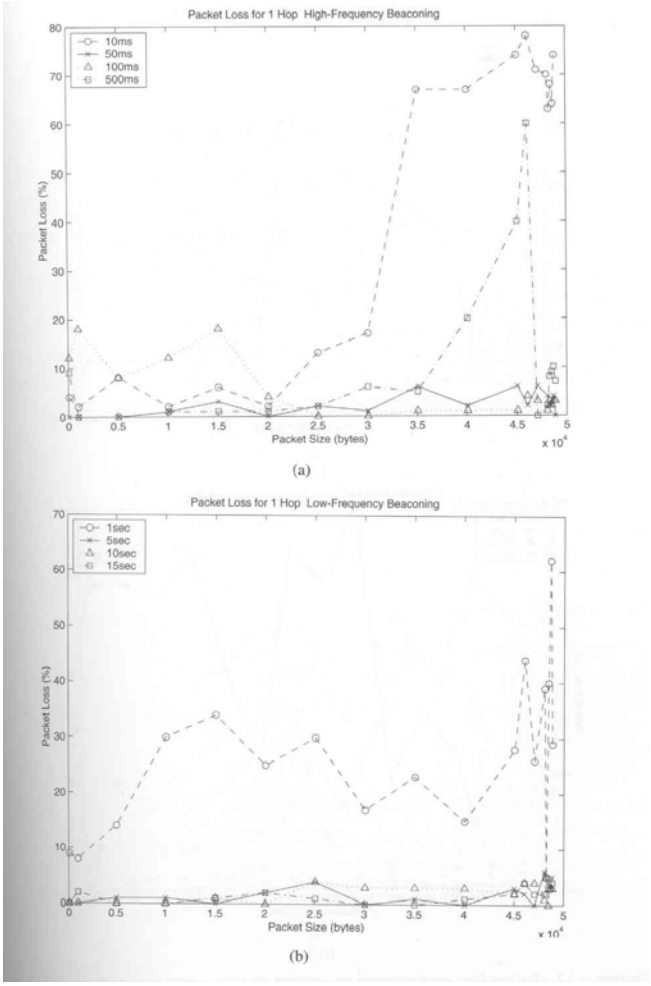

Figure 21. Packet loss performance vs. packet size at: (a) High frequency beaconing (b) Low frequency beaconing (From [2]).

5. Route Reconfiguration

Route reconstruction time is the time required for a host to reestablish a route when an intermediary route has moved. This route reconfiguration time does not vary with the increase in the number of hops. The beaconing interval and the packet size also did not have any effect on the route reconfiguration time.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. TOOLKIT DEVELOPMENT METHODOLOGY

A. INTRODUCTION

With the recent exponential growth in the wireless technology and the popularity of handheld devices there are abundant devices available in the market. The mobile devices that were chosen for implementation in this thesis were those that were readily available in the school. There are two major options available to develop the ad hoc toolkit. Option I was to use a Palm Device which ran on Palm OS and Option II was to use a Pocket PC device which ran on a Windows CE OS. In both the options the devices available were WI-FI enabled. The ad hoc network was to be built using the wireless 802.11 standard as this was the most widely accepted successfully implemented technology that supported the requirements of an ad hoc network. As was discussed in Chapter II the media access protocol requirements of an ad hoc network were satisfied to a great extent by the 802.11 protocol. The routing and other security requirements could be built on top of this protocol. The standard supported mobility within the radio range of the devices. The software requirements included the choice of an OS and then the choice of a development environment. The Microsoft Windows CE OS offered the choice of developing the toolkit using any one of the following environment:

i) Embedded Visual C++ 4.0

ii) Embedded Visual Basic

iii) .Net Compact Framework Environment

The .Net Compact Framework provided and integrated environment for developing, debugging, testing and deploying the embedded device software applications where as the other two environments catered for the development and debugging requirements but required extensive deployment procedures to be followed to run the applications developed. The .Net Compact framework also provided the flexibility of programming in either Visual Basic or C#.

The choice of Palm OS would have entailed the choice of a JAVA development environment. The resources available in this environment are extensive. The major compatibility issues are programs written for one OS are not supported by the other OS

and hence applications run only in one family of devices that run on the same OS. The other issue is some amount of backward incompatibility is also seen among the devices that run same OS.

For the implementation planned as part of this thesis I chose the .Net Environment for development and the Pocket PC environment for deployment. The implementation was tested using multiple Pocket PCs devices.

B. ARCHITECTURE

The topology of an ad hoc network keeps varying, and the performance of any control algorithm used in the network for purpose of scheduling transmission, routing and broadcasting require adapting to these changes efficiently. Topology management is achieved using either power control or hierarchical organization.

Power Control mechanisms base their control on power available on a per node basis so that one-hop neighbor connectivity is balanced and overall network connectivity is ensured. The hierarchical approach requires a subset of the network nodes to serve as the network backbone and support the control functions. This topology is called *Clustering* and consists of choosing *cluster heads* such that each node is associated with a cluster head. Once elected, the cluster heads are connected with one another directly or through gateways so that the union of cluster heads and *gateways* constitute a connected backbone. *Cluster heads* and *Gateways* reduce the complexity of maintaining topology information and simplify functions of routing, bandwidth allocation, channel access, power control or virtual – circuit support. For the clustering to be efficient and effective the *Cluster heads* and *Gateways* must be within radio range and as close as possible. The other important functionality is that the nodes in the network have to be aware of the other nodes that join the network or leave the network. Only then the nodes can elect cluster heads and gateways from among themselves.

This thesis has implemented a single hop network that consists of many nodes and a single cluster head which also acts as the gateway and can be used to connect to other gateways in the future as shown in figure 22.

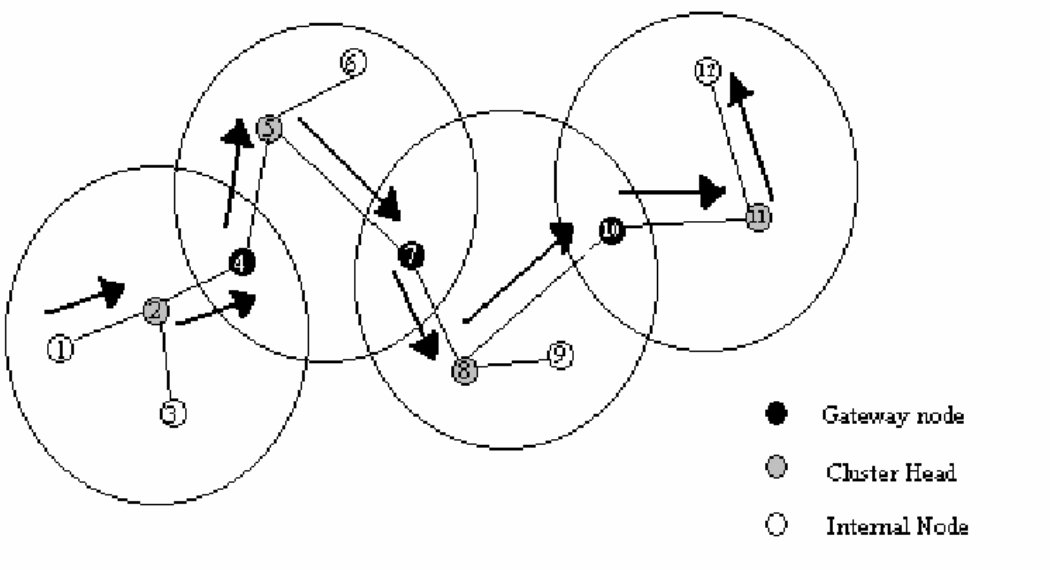

Figure 22. Routing from node 1 to node using CGSR (From [9])

1. Node Discovery Process

The node discovery process is implemented by the cluster head (Listener Agent) which listens for broadcasts in the network for a specific Communication ID. The network is assumed to consist of mobile nodes communicating through a common broadcast channel using omni-directional antennae with the same transmission range (in our case Pocket PCs WI-FI enabled.) The *Listener Agent* listens for *Clients* to register. Once registered the Clients are able to communicate to the Listener and also to other Clients registered with the Listener. The Listener is responsible to monitor the registration process of clients and also the de-registration process of clients. The Listener also ensures that messages received from a client are broadcast to every client registered with it. As the implementation consists of a single hop no routing is performed in this implementation.

2. Agent Module

The agent is required to perform the role of Listener or that of a Client as required and has to be built to support both the functionality. The functionality of the Listener requires the creation of two threads one to support the functionality of receiving

messages from the various clients and the other to transmit the messages received to other registered clients. The process of registration requires the creation of a unique communication ID which distinguishes messages sent within this network. The formation of this network would also entail the creation of some control messages to acknowledge receipts of message, process of registration, process of de-registration and transfer of control of the Listener to another node when the Listener wants to leave the network and transfers the control to another node. All this is achieved by creating a client that is capable of sending and receiving messages. The listener inherits all the functionality from the client and also implements additional functionality of distributing messages in the network. This is achieved by creating an additional thread running in the background which monitors a message queue to see if there is any message available for distribution. The node addresses to which the distribution is to be effected is stored in an array list of clients formed when clients register with the listener. The implementation uses the Multi Comm Framework (.dll file) developed by Jim Wilson, [6] as the implementation was found more responsive and effective when developed using the UDP datagram. The connectionless UDP datagram was more efficient in a wireless environment than the connection oriented TCP socket as it was difficult to maintain the connection in a mobile WI-FI environment and the connections had to be reestablished more than once to ensure transmission. The connectionless approach required that an additional retransmission mechanism be created whenever an acknowledgement was not received for all types of transmission. This was a small overhead and surmountable than overcoming problems to maintain connection in a connection oriented protocol.

3. Toolkit Functionality

The toolkit offers the following functionality:

a) Capability to “Start” a Collaborative chat Service – This option instantiates the Agent in a *Listener* Mode. This is the default mode and all agents are put into this mode when the software starts. In this mode the agent will:

   i. Listen for “Invite request” sent by various clients that want to join the chat environment or *conversation*. When this message is

received the “Invite” Message is sent to allow the client join the conversation.

ii. Send “Invite” messages to clients wishing to join the chat environment or *conversation*. When this message is sent the client acknowledges with an “Invite Accept” message that causes the registration of the client with Listener.

iii. Send “Terminate Conversation” message when the Listening agent wants to leave the conversation. Under this circumstance another client in the conversation can take up the function of a Listener or the chat service can be stopped. There can only be a single *Listener* (client operating in the Listening Mode) in every *conversation*.

iv. Forward messages received from one client to other clients registered with it.

b) Capability to “Join” a Collaborative chat Service - This option instantiates an agent in the *Client* Mode. This is not the default mode and the user has to choose this mode by a selection in the menu. In this mode the agent performs the following:

i. Send “Invite Request” Messages to join a conversation.

ii. Acknowledge “Invite” Messages to enable registration with a Listener.

iii. Send user generated *conversation messages* to the Listener.

iv. Send “Leave Conversation” message to indicate intention to leave the chat environment. This causes the listener to de- register the client from the conversation and will result in no further forwarding of messages from other clients.

c) Capability to “Modify” parameters for the chat environment - This option allows the user modify:

i. User Name – the name displayed with the message received.

ii. Client Port – The port on which the client sends and receives messages.

iii. Listener Port – The port on which the Listener sends and receives messages. (Note the agent when instantiated is either a client or a listener and ports have to be set as per the role the agent for successful communication.)

iv. Clear – Erase display area.

d) Capability to cut and paste parts of a conversation into a clipboard and store or view the clipboard.

e) Send a text file to all the clients in the conversation.

f) Capability to switch roles from that of a client to a listener and vice versa. However, it is to be noted that such switch over should take place only after all data transmissions are over so as to not cause any loss of data. Further any reversal of roles carried out should ensure the presence of only one listener in every conversation to ensure proper functioning.

4. Event Handling

All messages are processed by threads that run in the background. For efficiency, various events are generated to notify the application of different activities that have occurred. The following events are generated by the toolkit and require to be handled:

a) On Receive Event

b) On Join Event

c) On Leave Event

d) On Terminate Event

a.) "On Receive" Event – This event is to be handled by the listener. This event forms the heart of the program and is the mechanism used to notify the application that a message has been received. In this event, the listener receives a byte

array of data that has been sent by one of the clients. The received bytes are stored in an array called the “message pump” with the address of the sender. The messages stored in this message pump are then forwarded to all other clients, other than the sender, registered in the array called “client list”. There are three threads spawned by the listener when it is instantiated. One thread is used to send messages. The second thread is used to receive messages and the third thread is used to forward messages stored in the message pump. The client spawns only two threads one for sending messages and the other to receive messages.

b.) “On Join” Event - This event is handled by both the client and listener. For the client, this event conveys the conversation name and the IP address of the listener. For the listener, this event conveys the conversation name and the IP address of the client that has just joined in.

c.) “On Leave” Event – This event is handled by both the client and the listener. This event is fired by a specific client when it has left the conversation. For the listener this event conveys the conversation name and the IP address of the client who has just left the conversation. This event causes the removal of the IP address of the client from the “client list” and there by messages is not forwarded to this IP address in the future. For the client this event closes the agent and does a graceful exit from the application.

d.) “On Terminate” Event – This event is handled by both the client and the listener. This event indicates that the listener is terminating the conversation. Once this happens, no further messages from any client will be distributed. However,, if any one of the clients switch their roles and take up the role of a listener, a new conversation can be established.

5. Multithreading

The .NET compact framework is designed to be used in a client environment and therefore does not provide support for processor intensive, high performance applications and services which the full .NET framework supports. Therefore there are several threading features that the .NET compact framework does not support. Threads do not

have priorities in the .NET compact framework. If an application requires notification that a thread has terminated then code must be placed in the application that sets some synchronization mechanism such a Mutex before the thread exits and in the waiting thread, place code that performs a call to the wait-for-one method of the Mutex object. Suspend and resume methods which allow one thread to suspend and the other to resume respectively is not supported in the .NET compact framework. This has to be carried out by signaling a thread through a Mutex auto reset event or manual reset event. A background thread does not terminate when the owning application dies. Therefore the creation and manipulation of the background thread is not supported in the .NET compact framework. The creation and switching of control between threads running in the foreground and background is the primary requirement for the development of the chat application. The above-mentioned restrictions are overcome by the use of the .DLL file multi communication framework designed by [6]. Therefore the inclusion of the DLL file in our application provided us methods which could be used to create multiple threads running in the foreground and background simultaneously. This enables the application to become more responsive, scaleable and efficient.

VII. CONCLUSION

A. CONCLUSION

The potential for developing applications in a mobile ad hoc network is immense and the experience these applications will provide when deployed would entail them being termed as *smart* applications. These applications would be *aware* of their environment. Ad hoc mobile applications combine the characteristics of *neighbor-aware, location-aware, connectivity-aware and context-aware* smart applications in varying degrees and forms. For example a Context/Location aware application running on a wearable computer could detect the presence of a user in a shopping mall and inform the user of the bargains being offered, new product information and their availability to help him make informed decisions. Additionally online-catalogs, price comparisons, delivery information and other technical specifications can be received and shown to the user making his shopping experience virtually different. The scope and potential of such ad hoc applications is generally left to the readers imagination as of today as the protocols and technology that support ad hoc computing evolves. But it is only a matter of time when we will have such applications running of devices that are portable (as miniaturization improves), aware of other devices (as device discovery and routing protocols improves) and are context sensitive (using smart applications). Pervasive computing transcending physical boundaries and providing useful interactive environments can be formed using ad hoc network technologies. Motorola Piano Project and UC Berkeley Sensor Networks (Smart Dust Project) are next generation applications that are being studied in Laboratories.

Ad hoc wireless networking is evolving in both design of ad hoc routing protocols and prototype implementation. There are a large number of protocols which have been suggested and a few prototypes have also been implemented using the existing protocols like HTTP, TELNET, FTP and TCP/IP. These studies and experiments have provided varying degrees of results on throughput, EED and RD time and have thereby necessitated change in existing protocols to satisfy ad hoc environment needs. The changes may be minor in certain protocols (such as TCP/IP) and certain new protocols have to be created (such as SSDP, SLP V2). Implementing ad hoc wireless networks

using the current mobile computer technology, wireless adapters and Ad hoc routing software is feasible and the resulting communication performance is acceptable for most data application. However, Ad hoc routing protocols have to be fine tuned and mobile operating system modified to provide functionality to implement these ad hoc routing protocols. Currently ad hoc routing software is written from scratch as the existing mobile operating systems do not have the ad hoc routing functionality in built on them. The development of this toolkit will provide users the ability to test different quality of service requirements of an ad hoc network and also help him understand the functionality that would be required to be implemented in operating systems. The toolkit will also provide a mechanism which can be used to provide a collaborative service using the existing UDP protocol and multicast environment to provide for a chat service.

B. FOLLOW ON WORK

The first stage of the development of an ad hoc network would be creation of the network itself and testing applications that a based of a single hop. This would involve the creation of smart applications (User agents). This User Agent must then be aware of other devices in the neighborhood and form a *Directory* of User Agents. These User Agents must then be able collaborate and provide some *Service*. This thesis implemented the creation of an agent, development of a directory and establishment of a *Chat* and *File transfer facility* (Service) among the agents. This service was however limited to a single hop or a single group of User agents. Multiple groups of such agents must be able to communicate among themselves and that could be the next stage of this implementation. This would require the employment of any one of the routing mechanisms (protocols) discussed earlier and would require (Listener Agent) in this thesis provide an additional facility of communicating with other listeners in the vicinity using the routing mechanism devised.

APPENDIX. SOURCE CODE

```
using System;
using System.IO;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Text;
using MultiCommFramework;
using System.Net.Sockets;
using System.Net;


namespace MultiCommFramework
{
	/// <summary>
	/// Summary description for Agent.
	/// </summary>
	public class Agent : System.Windows.Forms.Form
	{
		private string _user = "Chat User";
		private string _conversation = "Multiple User";
		private bool _isHost = true;
		private string current = null;
		private int _clientPort = 5995;
		private int _listenerPort = 5998;

		private const string _originalTitle = "Client";
		private const string _hostTitle = "Host";
		private

		MultiCommClient _multiComm = null;
		Encoding _textEncoding = Encoding.UTF8;

	#region Windows Form generated code

		private System.Windows.Forms.TextBox txtInput;
		private System.Windows.Forms.TextBox txtDisplay;
		private System.Windows.Forms.MenuItem menuClient;
		private System.Windows.Forms.MenuItem menuLeave;
		private System.Windows.Forms.MenuItem menuModify;
		private System.Windows.Forms.MenuItem menuExit;
		private System.Windows.Forms.MenuItem menuSetting;
		private System.Windows.Forms.MenuItem menuJoin;
		private System.Windows.Forms.MenuItem menuUtility;
		private System.Windows.Forms.MenuItem menuSend;
		private System.Windows.Forms.MenuItem menuUserName;
		private System.Windows.Forms.MenuItem menuClientPort;
		private System.Windows.Forms.MenuItem menuListener;
		private System.Windows.Forms.MenuItem menuCurrentDir;
		private System.Windows.Forms.MenuItem menuTransmitFile;
		private System.Windows.Forms.MenuItem menuCopyText;
		private System.Windows.Forms.TextBox CopyBox;
		private System.Windows.Forms.MenuItem menuViewClipBoard;
```

```
private System.Windows.Forms.MenuItem menuSaveClipBoard;
private System.Windows.Forms.MenuItem menuCloseClipBoard;
private System.Windows.Forms.MenuItem menuclearDisplay;
private Microsoft.WindowsCE.Forms.InputPanel inputPanel2;
private System.Windows.Forms.MainMenu mainMenu1;

public Agent()
{
	//
	// Required for Windows Form Designer support
	//
	InitializeComponent();

	//
	// TODO: Add any constructor code after
	InitializeComponent call
	//
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
	base.Dispose( disposing );
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
this.mainMenu1 = new System.Windows.Forms.MainMenu();
this.menuSetting = new System.Windows.Forms.MenuItem();
this.menuModify = new System.Windows.Forms.MenuItem();
this.menuUserName = new System.Windows.Forms.MenuItem();
this.menuClientPort = new System.Windows.Forms.MenuItem();
this.menuListener = new System.Windows.Forms.MenuItem();
this.menuclearDisplay = new
System.Windows.Forms.MenuItem();
this.menuJoin = new System.Windows.Forms.MenuItem();
this.menuLeave = new System.Windows.Forms.MenuItem();
this.menuClient = new System.Windows.Forms.MenuItem();
this.menuExit = new System.Windows.Forms.MenuItem();
this.menuUtility = new System.Windows.Forms.MenuItem();
this.menuCurrentDir = new System.Windows.Forms.MenuItem();
this.menuTransmitFile = new
System.Windows.Forms.MenuItem();
this.menuCopyText = new System.Windows.Forms.MenuItem();
this.menuViewClipBoard = new
System.Windows.Forms.MenuItem();
this.menuSaveClipBoard = new
System.Windows.Forms.MenuItem();
```

```
this.menuCloseClipBoard = new
System.Windows.Forms.MenuItem();
this.menuSend = new System.Windows.Forms.MenuItem();
this.txtInput = new System.Windows.Forms.TextBox();
this.txtDisplay = new System.Windows.Forms.TextBox();
this.CopyBox = new System.Windows.Forms.TextBox();
this.inputPanel2 = new
Microsoft.WindowsCE.Forms.InputPanel();

        //
        // mainMenu1
        //
        this.mainMenu1.MenuItems.Add(this.menuSetting);
        this.mainMenu1.MenuItems.Add(this.menuUtility);
        this.mainMenu1.MenuItems.Add(this.menuSend);

        //
        // menuSetting
        //
        this.menuSetting.MenuItems.Add(this.menuModify);
        this.menuSetting.MenuItems.Add(this.menuJoin);
        this.menuSetting.MenuItems.Add(this.menuLeave);
        this.menuSetting.MenuItems.Add(this.menuClient);
        this.menuSetting.MenuItems.Add(this.menuExit);
        this.menuSetting.Text = "Setting";
        this.menuSetting.Click += new
        System.EventHandler(this.menuSetting_Click);

        //
        // menuModify
        //
        this.menuModify.MenuItems.Add(this.menuUserName);
        this.menuModify.MenuItems.Add(this.menuClientPort);
        this.menuModify.MenuItems.Add(this.menuListener);
        this.menuModify.MenuItems.Add(this.menuclearDisplay);
        this.menuModify.Text = "Modify";
        this.menuModify.Click += new
        System.EventHandler(this.menuModify_Click);

        //
        // menuUserName
        //
        this.menuUserName.Text = "UserName";
        this.menuUserName.Click += new
        System.EventHandler(this.menuUserName_Click);

        //
        // menuClientPort
        //
        this.menuClientPort.Text = "ClientPort";
        this.menuClientPort.Click += new
        System.EventHandler(this.menuClientPort_Click);


        //
        // menuListener
        //
```

```
            this.menuListener.Text = "ListenerPort";
            this.menuListener.Click += new
            System.EventHandler(this.menuListener_Click);

            //
            // menuclearDisplay
            //
            this.menuclearDisplay.Text = "ClearDisplay";
            this.menuclearDisplay.Click += new
            System.EventHandler(this.menuclearDisplay_Click);

            //
            // menuJoin
            //
            this.menuJoin.Text = "Join";
            this.menuJoin.Click += new
            System.EventHandler(this.menuJoin_Click);

            //
            // menuLeave
            //
            this.menuLeave.Text = "Leave";
            this.menuLeave.Click += new
            System.EventHandler(this.menuLeave_Click);

            //
            // menuClient
            //
            this.menuClient.Text = "Client";
            this.menuClient.Click += new
            System.EventHandler(this.menuClient_Click);

            //
            // menuExit
            //
            this.menuExit.Text = "Exit";
            this.menuExit.Click += new
            System.EventHandler(this.menuExit_Click);

            //
            // menuUtility
            //
this.menuUtility.MenuItems.Add(this.menuCurrentDir);

this.menuUtility.MenuItems.Add(this.menuTransmitFile);

this.menuUtility.MenuItems.Add(this.menuCopyText);

this.menuUtility.MenuItems.Add(this.menuViewClipBoard);

this.menuUtility.MenuItems.Add(this.menuSaveClipBoard);

this.menuUtility.MenuItems.Add(this.menuCloseClipBoard);

this.menuUtility.Text = "Utilities";
```

```
//
// menuCurrentDir
//
this.menuCurrentDir.Text = "CurrentDirList";
this.menuCurrentDir.Click += new
System.EventHandler(this.menuCurrentDir_Click_1);

//
// menuTransmitFile
//
this.menuTransmitFile.Text = "TransmitTextFile";
this.menuTransmitFile.Click += new
System.EventHandler(this.menuTransmitFile_Click);

//
// menuCopyText
//
this.menuCopyText.Text = "CopyText";
this.menuCopyText.Click += new
System.EventHandler(this.menuCopyText_Click);

//
// menuViewClipBoard
//
this.menuViewClipBoard.Text = "ViewClipBoard";
this.menuViewClipBoard.Click += new
System.EventHandler(this.menuViewClipBoard_Click_1);

//
// menuSaveClipBoard
//
this.menuSaveClipBoard.Text = "SaveClipBoard";
this.menuSaveClipBoard.Click += new
System.EventHandler(this.menuSaveClipBoard_Click);

//
// menuCloseClipBoard
//
this.menuCloseClipBoard.Text = "CloseClipBoard";
this.menuCloseClipBoard.Click += new
System.EventHandler(this.menuCloseClipBoard_Click);

//
// menuSend
//
this.menuSend.Text = "Send";
this.menuSend.Click += new
System.EventHandler(this.menuSend_Click);

//
// txtInput
//
this.txtInput.Location = new System.Drawing.Point(8,
8);
this.txtInput.Size = new System.Drawing.Size(224,
22);
```

```
            this.txtInput.Text = "";


            //
            // txtDisplay
            //
            this.txtDisplay.Location = new
            System.Drawing.Point(7, 44);
            this.txtDisplay.Multiline = true;
            this.txtDisplay.ReadOnly = true;
            this.txtDisplay.ScrollBars =
            System.Windows.Forms.ScrollBars.Both;
            this.txtDisplay.Size = new System.Drawing.Size(224,
            208);
            this.txtDisplay.Text = "";

            //
            // CopyBox
            //
            this.CopyBox.Location = new System.Drawing.Point(17,
            56);
            this.CopyBox.Multiline = true;
            this.CopyBox.ScrollBars =
            System.Windows.Forms.ScrollBars.Both;
            this.CopyBox.Size = new System.Drawing.Size(191,
            184);
            this.CopyBox.Text = "";
            this.CopyBox.Visible = false;

            //
            // Agent
            //
            this.ClientSize = new System.Drawing.Size(240, 264);
            this.Controls.Add(this.txtInput);
            this.Controls.Add(this.CopyBox);
            this.Controls.Add(this.txtDisplay);
            this.Menu = this.mainMenu1;
            this.Text = "Collab Agent";

      }
            #endregion

      /// <summary>
      /// The main entry point for the application.
      /// </summary>

      static void Main()
      {
            Application.Run(new Agent());
      }
            #endregion
```

```
#region MultiComm Management

            /// <summary>
            /// Create an instance of the MultiComm object.
            /// The exact object (Client or Listener) is determined
            /// by the Host choice in the option dialog.
            /// If host is checked, the Listener is created
            /// otherwise client
            /// The only difference from the program standpoint
            /// between client and listener is in the initialization,
            /// the rest of the program is unaffected
            /// </summary>
            private void CreateMultiComm()
            {
                  if (_isHost)
                        _multiComm = CreateMultiCommListener();
                  else
                        _multiComm = CreateMultiCommClient();

                  _multiComm.OnReceive += new
                  ReceiveEventHandler(_multiComm_OnReceive);
                  _multiComm.OnTerminate += new
                  TerminateEventHandler(_multiComm_OnTerminate);

            }

            /// <summary>
            /// Create client object
            /// </summary>
            /// <returns></returns>
            private MultiCommClient CreateMultiCommClient()
            {
                  MultiCommClient client = new
                  MultiCommClient(_conversation, _clientPort);

                  client.OnJoined += new
                  JoinedEventHandler(client_OnJoined);
                  client.Start();
                  client.SendInviteRequest(_listenerPort);
                  current = _originalTitle;

                  return client;
            }

            /// <summary>
            /// Create instance of a listener and attach to the
            /// relevent events
            /// Returns client because all ongoing behaivior is tracked
            as client
            /// </summary>
            /// <returns></returns>
            private MultiCommClient CreateMultiCommListener()
            {
```

```
            MultiCommListener listener = new
            MultiCommListener(_conversation, _clientPort,
            _listenerPort);

            listener.OnJoined += new
            JoinedEventHandler(listener_OnJoined);
            listener.OnLeave += new
            LeaveEventHandler(listener_OnLeave);
            listener.Start();
            listener.SendInvitation();
            current = _hostTitle;

            return listener;
      }

      /// <summary>
      /// Shutdown the current MultiComm object
      /// </summary>
      private void StopMultiComm()
      {
            if (_multiComm != null)
            {
                  if (! _isHost && _multiComm.IsInConversation)
                  {
                        SendMessage(_user + " has left");
                        System.Threading.Thread.Sleep(100);
                  // give message time to send
                  }
                  _multiComm.Stop();
                  _multiComm = null;
            }
            UpdateDisplay("You have left the conversation");
      }


            #endregion


#region MultiComm Event Handlers

      /// <summary>
      /// Fired by Listener (Host) when a client joins the
      conversation
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void listener_OnJoined(object sender,
      JoinedEventArgs e)
      {
      DoUpdateDisplayInvoke(string.Format("Connected: {0}",
      e.ipMember));
      }
```

```
        /// <summary>
        /// Fired by Listener (Host) when a client leaves the
        conversation
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void listener_OnLeave(object sender, LeaveEventArgs
        e)
        {
              DoUpdateDisplayInvoke(string.Format("Disconnected:
        {0}", e.ipMember));
        }

        /// <summary>
        /// Fired by client when the Listener (Host) acknowledges a
        /// request to join the conversation
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void client_OnJoined(object sender, JoinedEventArgs
        e)
        {
              DoUpdateDisplayInvoke("You have joined: " +
        _conversation);
              SendMessage(_user + " has joined");
        }

        /// <summary>
        /// Fired each time a conversation message is received
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void _multiComm_OnReceive(object sender,
        ReceiveEventArgs e)
        {
        DoUpdateDisplayInvoke(_textEncoding.GetString(e.dataBuffer,
        0, e.dataBuffer.Length));
        }

        /// <summary>
        /// Fired when the Listener (Host) terminates the
        conversation
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void _multiComm_OnTerminate(object sender,
        TerminateEventArgs e)
        {
        DoUpdateDisplayInvoke("Host has terminated conversation");
        }

#endregion
```

```
#region Message Sending

            /// <summary>
            /// Do Actual send of a message to conversation
            participents
            /// </summary>
            /// <param name="message"></param>
            private void SendMessage(string message)
            {
                  if (_multiComm != null)

      _multiComm.Send(_textEncoding.GetBytes(message));
                  else
                        MessageBox.Show("Must first join a conversation
            (Conversation/Join)");
            }

            /// <summary>
            /// Initiate message send
            /// </summary>
            /// <param name="sender"></param>
            /// <param name="e"></param>
            private void mnuSend_Click(object sender, System.EventArgs
            e)
            {
                  string message = _user + ": " + txtInput.Text;
                  UpdateDisplay(message);
                  SendMessage(message);
                  txtInput.Text = string.Empty;
            }

            #endregion

#region Display Management

            /// <summary>
            /// Add a message to txtChatDisplay
            /// Handle details of scrolling text into view
            /// This also triggers an evaluation of available menu
            selections.  This
            /// is a somewhat inefficient place to do it but simplifies
            the code by
            /// providing single point for modification for menu
            selections.
            /// </summary>
            /// <param name="displayText"></param>
            private void UpdateDisplay(string displayText)
            {
                  string linePrefix = txtDisplay.Text.Length == 0 ?
                  ">>" : "\r\n>>";
                  txtDisplay.Text += linePrefix + displayText;
                  PositionChatDisplay();

            }
```

```
//  Helpers to transfer control to UI thread for display
updates
EventHandler _updateDisplayInvoker = null;
string _displayText = null;

/// <summary>
/// Do the work to place the display text in a member
variable
/// then transfer control to the UI thread
/// </summary>
/// <param name="displayText"></param>
private void DoUpdateDisplayInvoke(string displayText)
{
      if (_updateDisplayInvoker == null)
            _updateDisplayInvoker = new
      EventHandler(UpdateDisplayInvoke);

      _displayText = displayText;
      txtDisplay.Invoke(_updateDisplayInvoker);
}

/// <summary>
/// Do the actual transfer to the UI thread then display
the message
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void UpdateDisplayInvoke(object sender, EventArgs
e)
{
      UpdateDisplay(_displayText);
}

/// <summary>
/// Place a help message in the txtChatInput and highlight
to be overwritten.
/// Useful to display help message w/o requireing extra
screen real estate
/// </summary>
/// <param name="displayText"></param>
public void DisplayHelpMessage(long t2,DateTime t3,string
t4)
{
      string linePref = txtDisplay.Text.Length == 0 ? ">>"
      : "\r\n>>";
      txtDisplay.Text += linePref+t2 + "  " +t3 +"  "+ t4;
      //txtDisplay.SelectAll();
      PositionChatDisplay();
}


/// <summary>
/// Do the details of scrolling a display message into view
/// </summary>
private void PositionChatDisplay()
{
      txtDisplay.Select(txtDisplay.Text.Length, 0);
```

```
            txtDisplay.ScrollToCaret();
        }

        /// <summary>
        /// Do the details of realigning the screen when the input
        panel (SIP)
        /// is shown/hiddent
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void inputPanel1_EnabledChanged(object sender,
        System.EventArgs e)
        {
            int screenBottom = inputPanel2.VisibleDesktop.Height;
            if (! inputPanel2.Enabled)
                screenBottom -= statBarAdjustment;

            txtInput.Top = screenBottom - txtInput.Height + 1;
            txtDisplay.Height = txtInput.Top + 1;

            PositionChatDisplay();
        }

        /// <summary>
        /// Adjustment for difference in VisibleDesktop property of
        InputPanel
        /// when InputPanel is not visible
        /// </summary>
        const int statBarAdjustment = 26;

        #endregion

#region Menu Handling

        /// <summary>
        /// Switch between client/listener.
        /// For a client, switches to listener, changes menuname
        client
        /// For a listener, switches to client, changes menuname
        listener
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void menuClient_Click(object sender,
        System.EventArgs e)
        {
            if (current == null)
            {
                if(menuClient.Text == "Client")
                {
                    _isHost = false;
                    menuClient.Text = "Host";
                }
                else
                {
                    _isHost = true;
                    menuClient.Text = "Client";
```

```
            }
        }
        else
        {
            string displayMessage1 = "You have to leave the
            conversation to change";
            UpdateDisplay(displayMessage1);
        }


}

/// <summary>
/// Leave the current conversation.
/// For a client, notifies listener then shutsdown
/// For a listener, notifies all clients then terminates
the conversation
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuLeave_Click(object sender,
System.EventArgs e)
{
    Cursor originalCursor = Cursor.Current;
    Cursor.Current = Cursors.WaitCursor ;

    if (_multiComm.IsInConversation)
        SendMessage(_user + " has left");
    string displayMessage = _multiComm.Mode ==
    MultiCommMode.Listener ? "Terminating conversation" :
        "Leaving conversation";

    StopMultiComm();
    current = null;
    UpdateDisplay(displayMessage);
    Cursor.Current = originalCursor;
}

/// <summary>
/// Join the current conversation.
/// For a client this will start the MultiCommClient and
then request
/// an invitation for the conversation
/// For a listener, will start a MultiCommListener (stored
as a client)
/// and then broadcast an invitation to join the
conversation
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuJoin_Click(object sender, System.EventArgs
e)
{
    Cursor originalCursor = Cursor.Current;
    Cursor.Current = Cursors.WaitCursor ;

    CreateMultiComm();
```

```
//    DisplayHelpMessage("<Type message here then tap
send>");

      string displayMessage = _multiComm.Mode ==
      MultiCommMode.Listener ? "Attempting to locate
      available clients..." :
            "Attempting to locate host...";
      UpdateDisplay(displayMessage);
      Cursor.Current = originalCursor;
}

/// <summary>
/// Terminate the MultiComm if running then exit
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuExit_Click(object sender, System.EventArgs
e)
{
      StopMultiComm();
      this.Close();
}


/// <summary>
/// To change username that is displayed with messages
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuUserName_Click(object sender,
System.EventArgs e)
{
            _user = txtInput.Text;
            txtInput.Text = string.Empty;
}

/// <summary>
/// To change client port number
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuClientPort_Click(object sender,
System.EventArgs e)
{
            _clientPort = GetPortValue(txtInput.Text);
            txtInput.Text = string.Empty;
}

/// <summary>
/// To change Listener Port Number
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuListener_Click(object sender,
System.EventArgs e)
{
            _listenerPort = GetPortValue(txtInput.Text);
```

```
            txtInput.Text = string.Empty;
}

/// <summary>
/// Helper method to extract the Port value from the
Options form
/// </summary>
/// <param name="portAsString"></param>
/// <returns></returns>
private int GetPortValue(string portAsString)
{
      int port = 0;
      if (portAsString.Trim() != string.Empty)
            port = int.Parse(portAsString);

      return port;
}

/// <summary>
/// Initiate message send
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuSend_Click(object sender, System.EventArgs
e)
{
      string message = _user + ": " + txtInput.Text;
      UpdateDisplay(message);
      SendMessage(message);
      txtInput.Text = string.Empty;
}


/// <summary>
/// Display current directory listing
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuCurrentDir_Click_1(object sender,
System.EventArgs e)
{
      DirList(this);
      txtInput.Text = string.Empty;
      PositionChatDisplay();
}


/// <summary>
/// To Transmit text files
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuTransmitFile_Click(object sender,
System.EventArgs e)
{
      OpenFileDialog dlg = new OpenFileDialog();
```

```
        dlg.Filter = "All Files (*.*)|*.*|Text Files
        (*.txt)|*.txt";
        dlg.InitialDirectory = @"\";

        if (dlg.ShowDialog() == DialogResult.OK)
        {
              this.txtInput.Text = dlg.FileName;
              MessageBox.Show(dlg.FileName);
              PositionChatDisplay();
              FileRead(this);
        }

        txtInput.Text = string.Empty;

  }

  /// <summary>
  /// Copy selected text in the display to clipboard
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void menuCopyText_Click(object sender,
  System.EventArgs e)
  {
        // Ensure that text is selected in the text box.
        if(txtDisplay.SelectedText != "")
              CopyBox.Text += txtDisplay.SelectedText;

  }

  /// <summary>
  /// To view contents of clipboard
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void menuViewClipBoard_Click_1(object sender,
  System.EventArgs e)
  {
        CopyBox.BringToFront();
        CopyBox.Visible = true;
  }



  /// <summary>
  /// To save contents of clipboard
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void menuSaveClipBoard_Click(object sender,
  System.EventArgs e)
  {

        SaveFileDialog dlg = new SaveFileDialog();
        dlg.Filter = "Text File|*.txt";
        dlg.FileName = "user.txt";
        dlg.InitialDirectory = "\\My Documents";
```

```
        if (dlg.ShowDialog() == DialogResult.OK)
        {
                MessageBox.Show(dlg.FileName);
                StreamWriter sw = new
                StreamWriter(dlg.FileName, false);
                sw.Write(this.CopyBox.Text);
                sw.Close();
        }

        txtInput.Text = string.Empty;
}

/// <summary>
/// To close clipboard
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuCloseClipBoard_Click(object sender,
System.EventArgs e)
{
        CopyBox.SendToBack();
        CopyBox.Visible = false;
}

/// <summary>
///
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuModify_Click(object sender,
System.EventArgs e)
{

}

/// <summary>
/// To clear display area
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuclearDisplay_Click(object sender,
System.EventArgs e)
{
        txtDisplay.Text = string.Empty;

}

/// <summary>
///
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuSetting_Click(object sender,
System.EventArgs e)
{
```

```
            }

            #endregion

#region Utilities

            /// <summary>
            /// Function to display contents of current directory
            /// </summary>
            /// <param name="sender"></param>

            public static void DirList(Agent sender)
            {
                  DirectoryInfo dir;
                  if(sender.txtInput.Text != string.Empty)
                        dir = new DirectoryInfo(sender.txtInput.Text);
                  else
                        dir = new DirectoryInfo(".");

                  try
                  {
                        foreach (FileInfo f in dir.GetFiles("*.*"))
                        {
                              String name = f.FullName;
                              long size = f.Length;
                              DateTime creationTime = f.CreationTime;
                              sender.DisplayHelpMessage(size,creationTi
                              me, name);
                        }
                  }
                  catch (Exception e)
                  {
                        sender.UpdateDisplay("The directory has no
                        files/could not be read:"+e);
                  }
            }


            /// <summary>
            ///  Function to read and transfer text file
            /// </summary>
            /// <param name="sender"></param>
            public static void FileRead(Agent sender)
            {
                  string fileName = sender.txtInput.Text;
                  try
                  {
                        // Create an instance of StreamReader to read
                        from a file.
                        // The using statement also closes the
                        StreamReader.
                        using (StreamReader sr = new
                        StreamReader(fileName))
                        {
                              String line;
                              // Read and display lines from the file
                              until the end of
```

```
                        // the file is reached.
                        while ((line = sr.ReadLine()) != null)
                        {
                              sender.SendMessage(line);
                              sender.UpdateDisplay(line);
                        }
                  }
            }
            catch (Exception e)
            {
                  // Let the user know what went wrong.
                  sender.UpdateDisplay("The file could not be
                  read:");
                  sender.UpdateDisplay(e.Message);
            }
      }

      /// <summary>
      /// Function to save contents from clipboard to a file
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      public static void FileWrite(Agent sender)
      {
            string path = @sender.txtInput.Text;
            // Delete the file if it exists.
            if (File.Exists(path))
            {
                  File.Delete(path);
            }

            // Create the file.
            using (FileStream fs = File.Create(path, 1048576))
            {
                  Byte[] info = new
                  UTF8Encoding(true).GetBytes(sender.CopyBox.Text
                  );
                  // Add some information to the file.
                  fs.Write(info, 0, info.Length);
            }
      }


      #endregion

   }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Jay Wrolstad. Mobile Infrastructure Market Poised for Growth. URL: http://www.wirelessnewsfactor.com/perl/story/17975.html, September 2004

[2] C.K Toh . Ad Hoc Mobile Wireless Networks. Protocols and Systems.

[3] Andy Wigley/Stephen Wheelwright. .Net Compact Framework Core Reference.

[4] IETF Internet Draft for DSR. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt.

[5] Ad Hoc On Demand Distance Vector (AODV) Routing (RFC 3561)

[6] Jim Wilson JW Hedgehog, Inc. MSDN - Developing Smart Device WiFi Applications with the .NET Compact Framework. http://msdn.microsoft.com/mobility/samples/default.aspx?pull=/library/en-us/dnnetcomp/html/multicommframework.asp, September 2004.

[7] Papers from Proceedings of The third ACM International Symposium on Mobile Ad Hoc Networking and Computing, Lausanne Switzerland.

[8] Papers from Proceeding of the fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Annapolis Maryland USA.

[9] **Padmini Misra**, misra@cse.ohio-state.edu . Routing Protocols for Ad Hoc Mobile Wireless Networks. http://www.cse.ohio-state.edu/~jain/cis788-99/ftp/adhoc_routing/index.html#CBRP, September 2004.

[10] C.K TOH. Wireless ATM and Ad Hoc Networks. Protocols and Architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr Peter Denning
   Department of Computer Sciences
   Naval Postgraduate School
   Monterey, California

4. Professor Alex Bordetsky
   Department of Information Sciences
   Naval Postgraduate School
   Monterey, California

5. Professor Gurminder Singh
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California